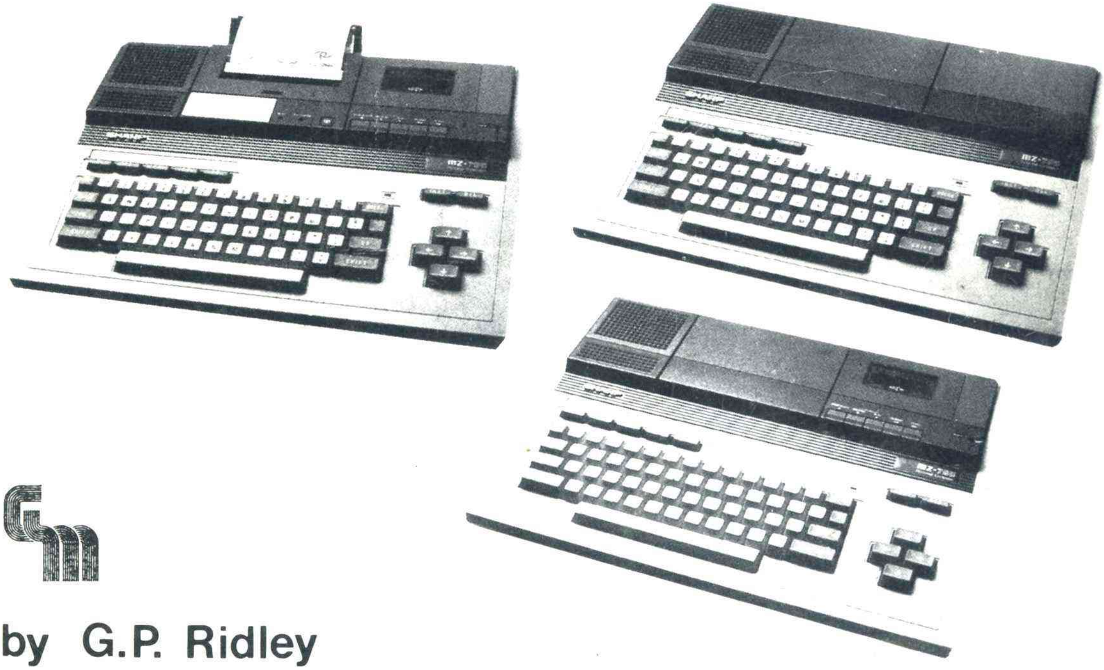
PEEKING AND POKEING

THE SHARP MZ-700 SERIES



by G.P. Ridley

A Guide to:
STYLING PROGRAMS
PEEKS AND POKES
VIDEO RAM
DATA STORAGE ON TAPE

PEEKING AND POKEING

THE

SHARP

MZ-700 SERIES

copyright © G.P. Ridley 1983

ALL RIGHTS RESERVED

No part of this publication may be reproduced by any means without the prior permission of the Author. The only exceptions are as provided for by the Copyright (photocopying) Act or in order to enter the software contained herein onto a computer for the sole use of the owner of this book.

First published 1983

PUBLISHED BY:

GM
D.C. BRENNAN Eng.
14 North Western Avenue
Watford, Herts. WD2 6AE

This book was produced on a BROTHER HR-I Daisywheel Printer fitted with a Centronics Interface using A4 paper and reproduced with a reduction of 20%.

FOREWORD

This book is written as a guide to understanding some of the lesser known programming aspects of the Sharp MZ-700 series Personal Computer.

The S-Basic used in programming the 700 is, in common with most micros, based on standard Microsoft Basic with a few variations. These will be explained along with the two, often misunderstood, commands PEEK and POKE.

Owners Manuals assume their readers possess a certain level of knowledge, but as is often the case one finds certain sections over simplified while others appear a complete mystery. I hope this book will clear some of the mystique and go a little way to giving you that all important leg-up on the ladder of understanding in regard to a most powerful Personal Micro, the MZ-700.

CONTENTS

1 ROM & RAM

The Memory Layout explained 7

2 Features of S-BASIC

2.1 Reserved Words 12
2.2 Tone Generator 16

2.3 Video Ram area

2.3.1 Character V-RAM 19
2.3.2 Colour V-RAM 23
2.3.3 V-RAM Additional characters ..24

3 Basic Additions

3.1 S-Basic Tape copy 33

3.2 Additions to Basic

3.2.1 Ready message 35
3.2.2 Trace Command 36
3.2.3 Slow running 37
3.2.4 Halt List 38
3.2.5 Bell on Ready 39
3.2.6 Buzz on Error 39
3.2.7 Data tape messages 41
3.2.8 Copy tape with additions 43

4 PEEKS & POKES

Peeks, Pokes, USR and CHR$ 46

5 Plotter Printer

5.1 Text Mode Commands 54
5.2 Graphic Mode Statements 55

6 Program Tips

6.1 Recovering lost programs 63
6.2 Converting MZ-80K to MZ-700 ..66
6.3 Logical Operators 67
6.4 User prompts 68
6.5 Delay Loops 68
6.6 User Breaks 68
6.7 GET Auto repeat 69
6.8 Colour changes 70
6.9 Protecting Programs 71
6.10 List extra 72

7 Data Handling

7.1 Sorting Data 73
7.2 Data tape handling 76

8 Interpreter Points

8.1 Basic Interpreter 84
8.2 RAM Monitor addresses 85
8.3 Basic Tokens and addresses ...86
8.4 Useful addresses 88
8.5 ROM Monitor addresses 90

Appendix

Display code table 2 92
Display code table 1 93
ASCII code table 94
Hex to Dec converter table 96

1. ROM & RAM

Some knowledge of how the Sharp micro stores data is helpful if one is to understand something about the two statements PEEK and POKE.

The memory of the MZ-700 series is compiled of a number of bytes, some are in ROM and the majority in RAM. ROM stands for Read Only Memory, and that is exactly all the micro can do with it, read and act upon the instructions contained within that area, the Monitor Program 1Z-013A which springs into life each time one switches on is contained in ROM, the size of this area is 4 Kbytes and is responsible for controlling any programs which are written in Machine Code or languages other than Basic.
After switching-on one loads the Basic tape, S-BASIC, this loads directly into RAM which stands for Random Access Memory, and is also known as Read/Write Memory, which is more accurate as unlike ROM the contents can be altered at will to suit ones needs.
The S-BASIC tape, known as the Basic Interpreter, loads into the RAM area and then switches out the area taken with ROM to give the user the maximum amount of memory for programs. On some other micros the Basic Interpreter is permanently in ROM, which means that the computer is ready to accept programs written in Basic immediately the micro is switched on. However the few minutes extra that it takes for the MZ-700 to load Basic into RAM is not derisory, as it enables us to alter Basic to our requirements very simply, and if one wishes to use a different language such as FORTH, PASCAL etc. one has not got the Basic Interpreter continually taking up precious memory space and being unused. Furthermore if one wishes to add some extra routines, similar to those in this book, one simply enters them from the keyboard, and if you wish to add a toolkit, which are tapes of additional commands in Basic, one only has to load the extra tape, whereas on micros with the Basic in ROM it can mean buying and replacing chips within the micro, at considerable extra cost.

The Memory Map of the MZ-700 is shown on pages 123/124 of the Owners Manual, and it should be noted that the numbers on the left side of the diagrams are in hexadecimal, that is numbers to the base of sixteen. The everyday numbers we use are decimal, to the base of ten. If one is going to progress to programming using Assembly Language or Machine Code then it will be neccessary to be acquainted with this system of numbering, nevertheless whenever a hex number is used in this book the decimal equivalent will be shown, there is also a conversion chart in the Appendix to assist you.
At this stage it should be pointed out that it is not neccessary to understand the Owners Manual from cover to cover to enable one to begin programming in Basic. Provided that you have read and understood the first two chapters on operating and the Basic commands it should be sufficient for you to get started. The Manual is a comprehensive guide to the MZ-700 from Basic through to the hardware configuration, and should appeal to experienced programmers and beginners alike. So if you have grasped most of the first two chapters and are bogged down trying to understand the Hardware or Monitor sections don't worry as it will not stop you writing Basic programs. Similarly with this chapter, if you find it heavy going don't despair you can always refer to it later, it is better to have more details on a subject than not enough, though one should read the next chapter on some of the Basic commands which may not be too clear from the Owners Manual.

MEMORY MAP

The first section of memory from address 0000 to 0FFF Hex (0 to 4095 Decimal) is where the Monitor 1Z-013A resides in ROM. Each byte is listed in the Owners Manual in chapter 5, it is essential to know the addresses of the Monitor subroutines if one is going to program in Assembly language, but not important in Basic.
RAM starts directly above this, and the area from 1000 to 11FFhex (4096 to 4607dec) is used to contain details of the program being used, such as size, execution address as can be seen on page 195 of

the Manual. This area is called the Monitor work area. Any Machine Language programs you write or buy will commence loading above this area in memory and will be controlled using the Monitor commands and routines described in chapter 5 of the Manual. At switch on the area from 1200 to CFFFhex (4608 to 53247dec) is completely empty and ready to use. The area starting at D000 to DFFFhex (53248 to 57343dec) is used to store current details of the screen, which is in RAM due to the screen details constantly changing, and is known as the Video Ram Area. Above this is the the Key Timer Ports and areas for controlling peripheral devices, such as printers and disc drives.

The Basic Interpreter tape commences loading directly above the Monitor work area at 1200hex (4608dec), and loads right up to 7DA0hex (32160dec). It's function is to interpret the lines of programs you write or load from tape into code that the computer understands, it interprets. It will recognise all the commands and statements which you are allowed to use in your programs, and will reject any incorrect ones, try entering GOTI instead of GOTO and you will see. After loading control is passed to location 7D79hex (32121dec) where a routine is located which swaps over the ROM Monitor and the RAM areas above D000hex and switches in RAM available to the user. It then moves the the complete Interpreter down to start at 0000hex, and the end becomes 6B79hex (27513dec). The Basic Interpreter contains it's own Monitor, we will call it the RAM Monitor to differentiate between the two. It contains routines to handle all the similar tasks that the ROM Monitor would normally do, but cannot because it has been swapped over, but they are all at different addresses, although the jump table at the beginning is similar (i.e. to call Bell one still enters USR(62) but it then jumps to a different address than used by the ROM Monitor). As with the ROM Monitor, Basic has to have work areas to store details of the current program, and there are several blocks of RAM throughout the Interpreter alocated as work areas. These areas of memory contain more information than the ROM work areas did, such as the position in memory that string variables are to be found. If your program has stored a variable such as A$, the computer must keep a

check on where it is stored in memory, so it can be recalled. No two progams will ever be the same length so the data will never be stored in the same place, the work area contains the address (pointer) to where they will be found in RAM. These areas are detailed more fully in chapter 8.
The area commencing at 6BCFhex (27599dec) is where Basic programs are stored from, either loaded from tape or typed in directly from the keyboard, they gradually grow upwards in memory reducing the amount of free memory remaining. Don't worry there's an awful lot of memory in the 700.

TOOLKITS

Toolkits, unlike their name suggests are not a mixture of screwdrivers, pliers etc., but tapes with extra commands which can be appended to S-BASIC to make the Interpreter even more powerful. Usually it will be possible to save the appended version onto one tape to make future loading simpler.

One such Toolkit is BAS-MOD which was specifically written for the MZ-700 series. Here are a few of the additional commands it contains:-

a) IF..THEN..ELSE...which is an improvement on the IF..THEN command which normally ignores any instruction after THEN if the IF argument is not true and the program would jump to the next line.

b) AND..OR..NOT...although it is possible to use AND and OR in S-BASIC by putting the arguments in brackets and using the + or - signs, NOT is not included, and it is more explicit to be able to enter them directly.

c) GOTO or GOSUB..LABEL...which allows one to specify by the LABEL statement the area where the program should jump to. It makes programming more explicit and debugging far easier as one the can enter GOSUB PRINT TITLES instead of GOSUB1200 and the subroutine will start with LABEL PRINT TITLES. If your program contains several jumps one will see at a glance where it is going.

d) **WHILE..WEND**...although similar to the FOR..NEXT loop it will only execute if the WHILE argument is true, FOR..NEXT will always execute once regardless of the argument.

e) **REPEAT..UNTIL**...similar again to FOR..NEXT except that the argument for repeating can be more complex. One could repeat UNTIL A=10 OR A<1 OR A>20. Both of these additions reduce the amount of extra lines one would have to write if only using FOR..NEXT loops.

The addition of a Toolkit should not reduce the amount of free memory that the user has, as most are written into the existing area taken up with the Basic Interpreter, one of the advantages of Basic being in RAM, as it can have extra code written to it.

2. Features of S-BASIC

2.1 RESERVED WORDS

Most of the Reserved Words or Commands in Basic V1.0A are detailed fully in the Owners Manual, but there are a few which may require further explanation.

LIMIT (abbreviated format LIM.)
This command is always either followed by a decimal number:-
LIMIT 49175
or a hexadecimal number preceeded by the $ sign:-
LIMIT $C017
or by a variable, already assigned a value, i.e. A=49157:-
LIMIT A

It is used to block off the upper part of RAM Memory for use by an assembly language routine. One reason for a Basic program containing such a routine can be to give greater effects to a games program for faster display on the screen, or another could be to include a routine which Basic does not contain and which the programmer added to enhance a program. Programs containing these routines execute far more quickly by using assembly code. You will not be able to write such routines without a knowledge of assembly language, but that will not stop you entering and running a program which you see listed in a magazine, provided it was written for the MZ-700. The first program on the Applications side of the Basic tape contains such a routine and the LIMIT is set to $9FFF in line 100, the assembly routine is READ from DATA statements at the end of the listing and POKEd into this blocked off area above $9FFF in lines 150-170. The routines are then called into operation by the USR($A000), which is one byte above the LIMIT previously set.
The reason for blocking off the top of memory in this way is because as a program is running certain data relating to it is stored from the top of memory downwards, if the top of memory is temporarily

lowered then the assembly routine will be safe from being overwritten during program execution.

LIMIT MAX (abbr. format LIM MAX.)
Must always be entered after running a program which contained a LIMIT statement, as entering NEW or LOAD will not clear the memory above the LIMIT set. If one is not sure if the previous program contained an assembly routine try entering PRINT SIZE and if a lower figure than 36588 is displayed then it probably did and LIMIT MAX should be entered to raise the top of memory to normal.

DIM
All variable arrays must be DIMensioned prior to use. Some Basics allow up to 10 subscripts without the use of DIM, but the MZ-700 does not, so however small the array DIM must be used first.

PEEK
This command allows the user to find the value that a particular byte contains, and returns the decimal value, it is used extensibly in programs which address the screen area directly within the Video Ram area, which is described later, to find which character is printed in a particular position on the screen.

POKE
Is used to alter the contents of memory, and must be used with care. It is also used when direct screen addressing to alter the display, or to enter assembly code routines into memory.
N.B. One does not need to enter the address for each byte if they are consecutive alterations. i.e. To print ABC at the beginning of the Video Ram area (at the top left corner of the screen) it is not necessary to enter POKE 53248,1:POKE 53249,2:POKE 53250,3
All that is needed is POKE 53248,1,2,3 for the same result.

CONSOLE (abbr. format CONS.)
Is used to print to certain sections of the screen area, leaving the remainder intact. It can be followed by either two or four parameters:- CONSOLE 5,15 or CONSOLE 5,15,10,20

or by variables already assigned a value:-

```
A=5:B=15:C=10:D=20
CONSOLE A,B,C,D
```

The first parameter is the starting line number, 0 being the top line. The second is NOT the last line number to be printed on, but the number of lines one wishes to be printed. So in the above case printing would commence on line 5 and stop on line 20 of the screen. Similarly with the third and fourth parameters, printing would start on column 10 and continue across the next 20 columns to position 30 across the screen, leaving whatever had been previously displayed around this area intact. While this command is in operation entering CLR or HOME will position the cursor at the top left corner of the specified area and NOT the top corner of the full screen, in order to achieve a complete clear screen one must enter CONSOLE with no parameters.

To illustrate enter the next one line program:-

```
10 FOR X=1 TO 400 :?X;:NEXT
```

And RUN

Now add line 5:-

```
5 CONSOLE 5,15,10,20
```

and RUN

Entering SHIFT/CLR will only clear the centre area.

N.B. The wrapping around of the screen effect described on page 129 of the Owners Manual by pressing SHIFT will either the cursor up or down key will only take effect if CONSOLE has been entered without parameters. Even if the largest print area of CONSOLE 0,25,0,40 has been specified the wrapping around will not take effect.

PRINT USING (abbr. format ?USI.)

Is an extremely useful addition to the MZ-700 extended Basic, there are however some points to be aware of.

If one has specified PRINT USING "###.##" and the number to be printed is larger that three digits before the decimal point, 'Over flow error' will be screened. If the number exceeds two digits after the point (i.e. 123.456) it will be either rounded up, in this case to 123.46, or rounded down if the last figure is less than 5. (i.e. 123.454 becomes 123.45)

PRINT USING " ££###.## " will print the £ immediately before the first digit, but specifying PRINT USING "£###.##" will print the £ always in the same column regardless of the size of the number to be printed. The TAB function can be included with this command to make monetary output exceptionally neat. i.e.

```
PRINT TAB(28);USING"£###.##";A
```

There are some Reserved words which are not shown in the manual but nevertheless still function.

CLS

Will perform a Clear Screen, either within a program or in direct mode. It does NOT need to be prefixed by PRINT.

HEX$

PRINT HEX$(49151) will convert the decimal number in brackets to hexadecimal, in this case BFFF would be printed.
One can convert from hex to decimal by entering PRINT$7000 and 28672 will be displayed. The highest hex number that can be converted is $8000, don't go higher it will be incorrect.

BOOT

Has the same effect as pressing the Reset button, and swaps in the ROM Monitor and passes control to it, at address 0000.

TRON/TROFF

When trying to debug a program entering TRON before RUN will display the correct line number being executed by the program, and is useful in tracing bugs. In its present form the line numbers are printed at the current cursor position, which is not too clear when there is a lot of printing to the screen taking place. In chapter 3 there is an alternative routine which can be added to S-BASIC which will display the line number in the top left-hand corner of the screen irrespective of the cursor position. TROFF will cancel the function.

2.2 TONE GENERATOR

The Tone Generator, which is responsible for the sometimes weird sounds heard on games programs, is dependant on the values contained in the two bytes OA39 and OA3A Hex (2617 and 2618 Dec). Values are POKEd into these addresses, usually within FOR..NEXT loops, and USR(68) turns the generator on and USR(71) turns it off. The best way of deciding which type of sound you might like to add to a program is by experimenting with different values POKEd into the two addresses.

Location 2618 controls the main pitch of the note and 2617 divisions within that note. If you enter the next short program and RUN, afterwhich we will alter it.

```
10 B=1
30 FOR A=1 TO 255
40 POKE 2617,A
50 POKE 2618,B
60 USR(68)
70 NEXT A
100 USR(71)
```

Now enter RUN

You should have heard a high note diminishing for just over a second.

If you add STEP 0.5 to the end of line 30 the duration will be 3 seconds.

The variable B has the value of 1, alter line 10 to:-

```
10 FOR B=1 TO 3
```

and line 70 to:-

```
70 NEXT A,B
```

and RUN

Now alter line 10 to:-

```
10 FOR B=1 TO 10
```

and RUN. You should notice that the higher the value of variable B becomes so the lower the note gets. Similarly the loops can be altered to decrement instead, which will cause the note to get higher in the scale.

Alter lines 10 and 30 to:-

```
10 FOR B=10 TO 1 STEP-1
30 FOR A=255 TO 1 STEP-10
```

and RUN

Altering the listing to the following will produce a sound similar to a Police siren. Change or add these lines:-

```
10 B=3
20 FOR C=1 TO 10
30 FOR A=255 TO 1 STEP-7
70 NEXT A,C
RUN
```

The loop in line 20 controlled the number of times it was heard. Experimenting with different values for A and B will produce a variation of sounds.

Here are a few examples which generate various sounds.

Enter NEW

```
10 FORA=1TO4
20 FORB=1TO150
30 POKE2618,B
40 USR(68)
50 NEXTB,A
60 USR(71)
```

and RUN. Now try:-

```
10 FORA=1TO6
20 FORB=1TO10
30 POKE2617,10-B
40 POKE2618,B
50 USR(68)
60 NEXTB,A
70 USR(71)
```

Using single statements on each line makes the listing more clear, but it is wasteful on memory as each line uses several extra bytes to store the line number and pointers to the position of where in memory the next line starts, so where possible make use of multi-statement lines such as on the next page.

Enter NEW and without spaces the next single line program. It should appear on screen like this:-

```
10FORA=255TO0STEP-12:POKE2617,A:POKE2618
,1:USR(68):NEXTA:MUSIC"R2":FORB=-255TO25
5STEP6:A=ABS(B):POKE2617,A:POKE2618,1:US
R(68):NEXTB:USR(71)
```

That should have simulated a Wolf-whistle.

Did you know that a single program line can be up to 255 characters long on the 700?

And now a Trimphone. Enter NEW and this one liner:-

```
10FORA=1TO6:FORB=1TO2:POKE2618,2:FORC=1T
O15:POKE2617,80:USR(68):FORD=1TO9:NEXTD:
POKE2617,255:USR(68):FORE=1TO9:NEXTE,C:M
USIC"R3":NEXTB:MUSIC"R7":USR(71):NEXTA
```

I love these one line programs.
Try altering the values for the different variables and compare the results. The MUSIC statement used above simply plays a note, in this case a Rest (silence), for a duration of 7 (1/2 a note, see page 66 of the Owners Manual).

If one omits to turn off the generator by not entering USR(71) the tone will continue, even after the READY message has been screened at the end of a program. Another use could be to attract the attention of the user when an error has occurred or when immediate input from the keyboard is required. The tone could continue until the error has been corrected or the correct key pressed. Afterwhich the tone could be turned off by USR(71) in the following line of the program.:-

```
200 POKE2618,2
210 USR(68)
220 GET A$:IF A$=""THEN220
230 USR(71)
```

2.3 VIDEO RAM AREA (V-RAM)

This area of memory stores the details of each individual location of the screen, in respect of characters and colours.

2.3.1 Character V-RAM

Usually any printing to the screen is achieved through using PRINT statements. While this is adequate for normal applications, it becomes too slow for games programs when printing is required in several directions on the display, in this case it is preferable to display the chosen character to be printed by directly POKEing it into the display area of memory. This part of memory is called the Video Ram area (V-RAM), and is responsible for storing the character each position of the screen area contains, it is described briefly on page 128 of the Owners Manual.

The screen is made up of 25 lines of 40 columns each, that is 1000 screen locations, that can be seen at any one time. The Character V-RAM area holds 2000 bytes which allows it to store two full screen displays, scrolling the display will show the complete 50 lines which make-up 2 screenfuls.

The address of the top left position of the screen is $D000 hex or 53248 decimal and the last screen location, bottom right, is $D3E7 or 54247 decimal. It is important to remember that however many lines are scrolled up or down the display always has position 53248 as the top left. To illustrate if one enters the 'CLR/SHIFT' keys to clear the screen and moves the cursor down the screen about halfway and enters POKE53288,26 and CR, the Z character is displayed on the left position of the second line of the screen. The top position is 53248 so the second line will be 40 positions higher at 53288. Enter ?PEEK(53288) CR and 26 will be printed as the value of the character at position 53288. Hold down the 'SHIFT' key and press the cursor-down key just once so that the 'Z' moves up one line to the top left position, now enter ?PEEK(53288) CR and 0 will be displayed as location 53288 now has nothing there, the 'Z' has moved into location 53248, top left. Entering ?PEEK(53248) CR will confirm this by displaying 26, which is now the value it contains. POKEing can be made to the area after $D3E7 (54247) but it will not be displayed until the screen is scrolled. If one clears the screen and enters POKE54287,26 which is 40 positions higher than the bottom right location of 54247, the letter 'Z' will not be visible until the screen has been scrolled up one line. Furthermore notice that POKEing to the screen area does not move the cursor, it remains where it was. It must also be noted that all POKEing to this area one must use the Display Code table and NOT the ASCII code table. It does not matter whether one uses Hex or Decimal numbers when POKEing and PEEKing the screen area, one can mix them in the same statement if desired. The following statements are all valid and they all print the letter Z in the top left screen position:-

POKE $D000,$1A   POKE $D000,26   POKE 53248,$1A   POKE 53248,26

The Display Code table in the Manual has the values in Hex, but if one prefers working in decimal the table has been re-printed in the Appendix of this book with the decimal values, along with the ASCII Code table.

To demonstrate here is a simple program where the figure of a man is falling down the screen and then moving off to the right. Do not enter the REMarks they are for explanation only.

```
10 PRINT"■"         :REM CLEAR SCREEN
20 TL=53248         :REM TL=TOP LEFT OF SCREEN=53248
30 P=0              :REM P=Position Counter
40 GOSUB140         :REM SUBROUTINE TO POKE MAN INTO NEW POSITION
50 POKE TL+P,0      :REM ERASE MAN FROM OLD POSITION
60 IFP=960THEN90    :REM DON'T PRINT DOWN FURTHER THAN THIS
70 P=P+40           :REM ADD 1 LINE TO OLD POSITION (40=1 COMPLETE LINE)
80 GOTO40           :REM GO BACK AND PRINT AGAIN
90 P=P+1            :REM MAN ON BOTTOM LINE. ADD 1 TO P TO MOVE RIGHT
100 GOSUB140        :REM GO AND PRINT IN NEW POSITION
110 IFP=999THENEND  :REM PRINT NO FURTHER
120 POKE TL+P,0     :REM ERASE FROM OLD POSITION
130 GOTO90          :REM GO BACK AND PRINT AGAIN
140 POKE TL+P,202   :REM SUBRTNE TO POKE MAN (202) INTO NEW POSITION
150 FORDL=1TO200    :REM DELAY LOOP. TO SPEED-UP MAKE 200 LESS
160 NEXTDL
170 RETURN
```

One can see that the location to POKE to can be a variable, already assigned a value, or a calculation such as line 50. Also on completion the cursor has remained at the top of the screen.
If lines 150/160 are deleted one will see just how fast POKEing to the screen really is.

This next listing shows how one can move a character around the screen by calculating the position to move to. The man is originally placed at position 53748 in line 20 which, as the screen has 1000 positions adding 500 to the top left position of 53248, puts it in the centre of the screen.
The GET statement is used to find which way we want the man to move using the keys:-

Q=north-west W=north E=north-east
A=west D=east
Z=south-west X=south C=south-east

Lines 60-130, depending on the key entered, decide which position to re-print the man. If W is pressed then it must be moved one line upwards, which is 40 positions lower in memory, so 40 must be subtracted from variable P for it's new position. Similarly if X is pressed then it must move down one line and 40 is added to P. The moves to East and West are probably the easiest to calculate as the man either moves one position left or right, so P would have either 1 subtracted or added to it. It's the diagonal moves which require a little thought, pressing Q for the direction of North-West means subtracting 41 from P, and W for North-East means subtracting 39 as one line directly above the old position is -40, so either side of this becomes -39 or -41. And the same applies for South-West and South-East only this time we add either 39 or 41.
The variable PP in line 50 is the previous position of the man which, if one of the only 8 keys the program searches for has been pressed, must be POKEd with zero to erase it as if it wasn't the screen would soon be plastered with little men. Line 150 checks if the man is either at the top or the bottom of the screen by testing P to see if it is smaller or larger than the screen parameters, and if so then the program jumps back to the GET line. This line is how

the 700 executes the OR statement, by putting both arguments within parentheses coupled with the + sign, it is explained more fully in the chapter on Program Tips.
Enter NEW and CR.

```
10 PRINT"■"
20 P=53748
30 POKEP,202
40 GETA$:IFA$=""THEN40
50 PP=P
60 IFA$="W"THENP=P-40:GOTO150
70 IFA$="Q"THENP=P-41:GOTO150
80 IFA$="A"THENP=P-1:GOTO150
90 IFA$="Z"THENP=P+39:GOTO150
100 IFA$="X"THENP=P+40:GOTO150
110 IFA$="C"THENP=P+41:GOTO150
120 IFA$="D"THENP=P+1:GOTO150
130 IFA$="E"THENP=P-39:GOTO150
140 GOTO40
150 IF(P<53248)+(P>54247)THENP=PP:GOTO40
160 POKEPP,0
170 GOTO30
```

Locations can be checked prior to moving to them to see what they contain by using the PEEK statement. If one adds these lines it will be seen that the man will only move if the position is empty:-

```
11 FORX=1TO100
12 B=INT(RND(1)*999)+1
13 POKE53248+B,83
14 NEXT
155 IFPEEK(P)<>0THENP=PP:GOTO40
```

Line 12 randomly selects values between 1 and 1000 for variable B. Line 13 POKEs the start of V-RAM (53248) plus B with the Heart character, and line 14 loops back for 100 times. This will print Heart signs at random over the whole screen area. Line 155 checks that the location to move to is empty by PEEKing it, and if it contains a character the man will not move and the program will go back to the GET statement. There are further listings using POKEs to the screen area at the end of this chapter.

2.3.2 Colour Data V-RAM

It follows that if each position of the screen can store it's contents somewhere in memory that it also must store it's relative colour co-ordinates too. Whereas the characters were stored from $D000 onwards the colours for each screen location start at $D800 hex or 55296 decimal which is the top left position on screen and the bottom right location is $DBE7 or 56295. As before the area is two screenfuls of 50 lines by 40 columns.
To discover what character is at the top left of the screen one would enter ?PEEK(53248), and to find the colour of the same position one would enter ?PEEK(55296). It would be ideal if these starting locations were round figures, but that would be too easy. A good tip is that the Colour V-RAM is always 2048 higher than it's equivalent Character V-RAM location. Obviously each screen position can be allocated a different colour combination, whether it is by the COLOR or the POKE statements. It must be remembered that if one PEEKs a location to find out what character it contains the decimal value will be returned, and it is the same with the Color V-Ram which can at first sight cause confusion.

If the screen is printing white characters on a blue background any location PEEKed within 55296 and 56295 will return the value 113, this is because each hex number contains two figures, from 00 to FF hex, and in the case of the Color V-Ram the first of those figures represents the character colour and the second the background colour. Hence with the colours set at white on blue background the hex value will be 71, 7 being white and 1 being blue, but this is hex not decimal so when PEEKed the decimal equivalent is printed, which is of course 113. There is a conversion table in the Appendix of this book. Similarly if the screen was printing blue on a light blue background the value returned for ?PEEK(55296) would be 21, which is the decimal of 15 hex (1=blue and 5=light blue). Obviously if one wants to POKE a position with a different colour it is easier as you can enter the hex number anyway, POKE55296,$62 will alter the top left screen position to printing yellow on a red background, which is easier than converting to decimal each time.

2.3.3 V-RAM Additional characters

Virtually hidden within your MZ-700 are an additional 255 characters to add to those already in the Display Code table. The Japanese version of the 700 does not contain them, that is probably why they aren't mentioned in the Owners Manual. The characters, which are re-printed in the Appendix of this book, are POKEd onto the screen in the same way the standard set of Display code characters are but first certain changes have to be made to the contents of the addresses of the Colour V-RAM area.

To re-cap on the Colour V-RAM it must be remembered that the highest value a colour can be is 7 (white), therefore the highest value a colour co-ordinate of the screen should ever be is 77 hex, which would be printing white characters on a white background, which would be useless anyway as one could not see it, but nevertheless normally it shouldn't hold a higher value. The second or alternate set of Display codes are selected by setting bit 7 of the colour V-RAM address to 1. For those of you who are yet uninitiated into the mysteries of Assembly language this is equal to adding 80 hex or 128 decimal to the value of the colour stored at that location of the screen.
Regardless of the current colours enter COLOR,,7,1 and 'CR' and then SHIFT/CLR. We know that the screen is printing white on blue so all locations within the Colour V-RAM area will contain 71 hex (113 dec). With the cursor on the top line press the 'ABC' keys without 'CR' key and move the cursor down screen and back to the leftmost column by the cursor keys. If one alters the top left hand Colour V-RAM position ($D800 or 55296) from 71 hex to F1 hex that square will alter from displaying the character from the first Display set to the alternative set. In this case the alternate equivalent is 'A' in outline form. Enter POKE55296,$F1 and 'CR', and the 'A' has changed. If you wish to stay in decimals remember the current colour is 113, so adding 128 it becomes 241, therefore the above could be entered as POKE55296,241, for the same result. To alter all three letters on the top line enter POKE55296,241,241,241 and 'CR'. That is a pleasant addition in that if consecutive bytes are being altered one

does not need to nominate the address for each POKE, only the first address.

One can also turn certain areas of the screen on, so that the characters POKEd to those positions will subsequently be of the second Display code, but one must first work out their addresses correctly. Taking the top square in the Character V-RAM as 53248, the tenth line down will commence at 53248+10*40=53648. In order to display at this position a character from the second character set the equivalent location in the Colour V-RAM must be calculated in the same way. The top square in Colour RAM is 55296, similarly line ten will start at 55296+10*40=55696, so this location must have 80 hex (128 dec) added to it's current value, now all POKEs to 53648 will be in the second character set. The Hangman program uses this method as will be seen at the end of this chapter.

The Display code tables have similarities in that the capitals A to Z and the numerals 0 to 9 have the same positions, therefore a screen message using capitals can be displayed using the second (outline) character set. The byte which stores the current colours to print is location 005D hex (93 dec), at the moment it should be 71 hex (113 dec), adding 80 hex (128 dec) to it's present contents will display the next PRINT statement in the second character set. Enter:-

```
POKE$5D,PEEK($5D)+$80:PRINT"SHARP"
```

This is handy for highlighting a message, furthermore on completion location 005D returns to it's former value so any subsequent PRINTs will be normal. The above line is simply PEEKing location 005D and adding 80 hex to it. No matter what colours were being printed at that time they would remain the same. It could also be entered as:-

```
POKE93,PEEK(93)+128:PRINT"SHARP"
```

If a games program is purely using characters from the second character set then the Colour V-RAM display will need to be altered throughout. Once the colours have been decided this can be done with a FOR..NEXT loop. If it was to be white on blue a line could be entered:-

```
FOR X=55296 TO 56295:POKEX,241:NEXT
```

And all characters POKEd to the Character V-RAM area (53248 to 54247) will be in the second Display Code character set. In these examples we have only dealt with one screenful of locations, and it should be remembered that both Character and Colour V-RAM hold an additional 1000 locations, round the back of the display, and these can be altered too for greater effects when scrolling up or down takes place.

Character V-RAM is from D000 (53248) to D7CF (55247) and Colour V-RAM from D800 (55296) to DFCF (57295) making 2000 bytes in each.

The following charts are for converting the colours to display the second character set.

DECIMAL					CHARACTERS				
B		Black	Blue	Red	Purple	Green	L.Blue	Yellow	White
A	Black	128	144	160	176	192	208	224	240
C	Blue	129	145	161	177	193	209	225	241
K	Red	130	146	162	178	194	210	226	242
G	Purple	131	147	163	179	195	211	227	243
R	Green	132	148	164	180	196	212	228	244
O	L.Blue	133	149	165	181	197	213	229	245
U	Yellow	134	150	166	182	198	214	230	246
N	White	135	151	167	183	199	215	231	247
D									

HEXADECIMAL					CHARACTERS				
B		Black	Blue	Red	Purple	Green	L.Blue	Yellow	White
A	Black	80	90	A0	B0	C0	D0	E0	F0
C	Blue	81	91	A1	B1	C1	D1	E1	F1
K	Red	82	92	A2	B2	C2	D2	E2	F2
G	Purple	83	93	A3	B3	C3	D3	E3	F3
R	Green	84	94	A4	B4	C4	D4	E4	F4
O	L.Blue	85	95	A5	B5	C5	D5	E5	F5
U	Yellow	86	96	A6	B6	C6	D6	E6	F6
N	White	87	97	A7	B7	C7	D7	E7	F7
D									

HANGMAN

The unused letters of the alphabet are printed on the third line of the screen, in fact the letter 'A' is printed in position 53338 of the Character V-RAM area. This position is calculated by remembering that the top left position is 53248, so the start of line two is 40 positions higher at 53288, likewise the third line commences at 53328, 40 higher still. The letter 'A' is printed 10 places in from the left, so it's position on screen is 53328+10=53338. When you run the program you will notice that the letter selected moves from the unused to the used line. We know, by looking up the decimal ASCII code value in the Appendix, that the letter 'A' has a value of 65. If we deduct 65 from the position of 'A' (53338) we get 53273, so in program line 300 we POKE 53273 plus the ASCII value of the letter entered. If it was the letter 'A' it would be 53273+65 making 53338, so we POKE 53338,0 which places a space where 'A' was in the unused line. In order to re-print it two lines lower on screen, in the used section, we add 80 to the original marker of 53273, making 53353 and once again add the ASCII value to this figure. Here a slight problem is encountered as although the position is calculated quite easily, the character we wish to POKE into this position cannot be done by using the ASCII code, we must use the Display code which as one will see have different values. Luckily the letters A-Z are always 64 lower in Display code than there ASCII equivalent, therefore in line 320 we POKE the calculated position with the ASCII value minus 64 to display the equal letter. The head on the gallows is from the second Display code set and these four locations are turned on to that set in line 150 by POKEing them with 241, which maintains the white on blue colours but means all POKEing to these squares will be in the second set of characters. The drawing of the gallows is calculated from a seed position of 54078 in line 550 and then drawn by READing DATA statements which contain, in groups of two, the related position from 54078 and the actual display code of the character to be displayed. Line 450 prints at 54078+40 character 112, followed by 54078+41 also character 112, which will print them next to each other in a straight line. One can also see how the face changes in lines 510-520 as the trap opens.

```
10 COLOR,,7,1
20 TEMPO7
30 DIMA$(26),H$(26)
40 PRINT"■";:PRINT[1,7]"■■ ENTER YOUR WORD (don't let anyone see)"
50 PRINT[2,7]TAB(8); " 26 Letters is Maximum "
60 PRINT
70 INPUTY$
80 B=LEN(Y$)
90 FORX=1TOB:A$(X)=MID$(Y$,X,1)
100 IF(ASC(A$(X))<65)*(ASC(A$(X))>90)THENGOTO120
110 NEXTX:GOTO140
120 PRINT[7,2]"WORD MUST ONLY CONTAIN A to Z"
130 MUSIC"-B7RC7":GOTO40
140 PRINT"■";:PRINT[1,7]TAB(7);" THE WORD HAS";B;" LETTERS "
150 FORX=55929TO55969STEP40:POKEX,241,241:NEXT
160 PRINT"■UNUSED:-■■ABCDEFGHIJKLMNOPQRSTUVWXYZ"
170 PRINT:PRINT"■■USED:-"
180 CURSOR6,8:FORX=1TOB:PRINT"-";:NEXTX
190 G=1:T=0
200 CURSOR2,10:PRINT[7,1]"PLEASE PRESS LETTER No";G;"  ■■";
210 GETN$:IFN$=""THEN210
220 IF(ASC(N$)<65)+(ASC(N$)>90)THEN210
230 FORX=1TOLEN(AG$)
240 IFN$=MID$(AG$,X,1)THEN270
250 NEXTX
260 GOTO290
270 CURSOR2,10:PRINT[0,7]"THAT LETTER HAS BEEN TRIED"
280 FORX=1TO10:USR(62):NEXT:GOTO200
290 AG$=AG$+N$
300 PRINT[7,1]N$:POKE53273+ASC(N$),0
310 POKE55401+ASC(N$),242
320 POKE53353+ASC(N$),ASC(N$)-64
330 GOSUB400
340 IFV=0THENGOTO360
350 MUSIC"+#A+A":GOTO380
360 GOSUB550
370 IFL=99THEN600
380 IFT=BTHEN590
390 G=G+1:GOTO200
400 V=0
410 FORX=1TOB
420 IFH$(X)=N$THENV=1:GOTO440
```

```
430 IFN$=A$(X)THENCURSOR0,8:PRINTTAB(X+5);N$:V=V+1:T=T+1:H$(X)=N$
440 NEXTX:RETURN
450 DATA40,112,41,112,42,112,43,112,44,112
460 DATA-1,61,-41,61,-81,61,-121,61,-161,61
470 DATA-201,61,-241,61,-281,61,-320,60,-280,118
480 DATA-319,60,-318,60,-317,60 ,-276,113,-236,113
490 DATA-197,145,-196,146,-157,149,-156,150,-116,0
500 DATA-117,228,-116,230,-77,238,-76,238,-75,0
510 DATA-197,147,-196,148,-157,151,-156,152,-117,227
520 DATA-197,147,-196,148,-157,153,-156,154,-116,227
530 DATA-157,155,-156,156,42,115,43,0,44,61
540 DATA44,0.45,113,0,99
550 Z=54078
560 FORX=1TO5:READK,L
570 IFL=99THENRETURN
580 POKEZ+K,L:NEXTX:MUSIC"-G1":RETURN
590 CURSOR0,10:PRINT[1,7]" WELL DONE..THAT TOOK";G;" GUESSES ":GOTO690
600 POKE55929,113,113:POKE55969,113,113
610 FORX=53881TO54161STEP40:POKEX,0,113:NEXT
620 B=2
630 FORA=1TO240STEP2
640 POKE2617,A:POKE2618,B
650 USR(68)
660 NEXTA
670 USR(71)
680 CURSOR0,11:PRINT[1,7]"NO IT WAS ";Y$
690 PRINT[7,2]"■ ANOTHER GO? (Y/N) "
700 MUSIC"A5B5+C5 "
710 GETZ$:IFZ$=""THEN710
720 IFZ$="N"THENEND
730 IFZ$="Y"THENRUN
740 GOTO710
```

The next listing 'Connect 4' is for two players each trying to get four counters in a row. It also uses direct screen addressing to the V-RAM area to draw the grid in lines 200-320. As the counters stop on every other line, when checking for a win, the positions above and below the last counter must be in multiples of 80 as in the DATA statements in lines 800-820, where checking below it's position would mean PEEKing 80 for 1 line, 160 for 2 lines etc. Similarly adjacent columns being 2 squares apart are 2,4 and 6 as in line 830.

```
10 GOSUB1190
20 DIMNA$(2),C(9),X(9),B(3),FW(4),S(2)
30 FORN=1TO2
40 X=53858
50 PRINT"▣";:PRINT[1,7 ]" WHAT IS YOUR NAME PLAYER";N;" ? ":USR(62):PRIN
60 INPUTNA$(N)
70 N$="HELLO@"+NA$(N)
80 FORL=1TOLEN(N$)
90 A=ASC(MID$(N$,L,1))
100 P=X+L-20*(A-64)
110 M=X+L
120 POKEM+80,A-64
130 FORJ=MTOPSTEP-40
140 IFJ=PTHENPOKEJ,248:GOTO160
150 POKEJ,127
160 NEXTJ,L
170 USR(62):FORDL=1TO3000:NEXTDL
180 NEXTN
190 N=1
200 RESTORE
210 PRINT"▣"
220 FORA=1TO9
230 READX
240 FORB=1TO8
250 POKEX,121
260 X=X+40
270 POKEX,189
280 X=X+40
290 NEXTB
300 NEXTA
310 DATA53426,53428,53430,53432,53434,53436,53438,53440,53442
320 DATA55474,55476,55478,55480,55482,55484,55486,55488,55490
330 RESTORE
340 FORA=1TO8
350 READX
360 FORB=1TO8
370 POKEX+41,120
380 X=X+80
390 NEXTB,A
400 RESTORE
410 AA=33
420 FORA=1TO8
430 READX
440 POKEX+1929,$D1:POKEX-119,AA
```

```
450 AA=AA+1
460 NEXTA
470 CURSOR19,20:PRINT[1,7]"1 2 3 4 5 6 7 8"
480 PRINT"■SCORE"
490 CURSOR0,1:PRINTS(1)
500 CURSOR0,3:PRINTS(2)
510 CURSOR3,1:PRINTNA$(1);" IS ●"
520 CURSOR3,3:PRINTNA$(2);" IS 0"
530 CURSOR0,22:PRINT SPC(30)
540 P=N+70
550 CURSOR0,11:PRINT"WHICH COLUMN(1to8)"
560 PRINT:PRINTSPC(18)
570 PRINT"↓→→→";NA$(N):USR(62)
580 GETA$:IFA$=""THEN580
590 IF(ASC(A$)<49)+(ASC(A$)>56)THENGOTO580
600 PRINT"↑↑↑         COLUMN ";A$;"      "
610 PRINT:PRINTSPC(18)
620 RESTORE
630 FORA=1TO9
640 READX(A):NEXTA
650 FORA=1TO9:READC(A):NEXTA
660 AA=VAL(A$)
670 X=X(AA)-79:C=C(AA)-79
680 IFPEEK(X+80)<>0THENCURSOR8,22:PRINT[7,2]" COLUMN IS FULL ":GOTO700
690 GOTO710
700 MUSIC"-G2-C2":FORDL=1TO500:NEXT:GOTO530
710 POKEX,P:POKEC,$51
720 FORDL=1TO100:NEXT
730 IFPEEK(X+80)=0THENPOKEX,0:POKEC,$71:X=X+80:C=C+80:GOTO710
740 GOSUB780
750 IFTL=4THEN970
760 IFN=1THENN=2:GOTO530
770 N=1:GOTO530
780 CW=X:TL=1
790 FW(TL)=CW
800 DATA80,160,240,-80,-160,-240
810 DATA78,156,234,-78,-156,-234
820 DATA82,164,246,-82,-164,-246
830 DATA2,4,6,-2,-4,-6
840 FORV=1TO8
850 IFINT(V/2)<>V/2THENTL=1
860 FORA=1TO3
870 READB(A)
880 NEXTA
```

```
890 FORA=1TO3
900 IFPEEK(CW+B(A))<>PTHEN950
910 TL=TL+1
920 FW(TL)=CW+B(A)
930 IFTL=4THENRETURN
940 NEXTA
950 NEXTV
960 RETURN
970 CURSOR0,15:PRINT"THE WINNER IS"
980 CURSOR4,17:PRINTNA$(N)
990 S(N)=S(N)+1
1000 F=0
1010 Q=74
1020 IFF>=10THEN1100
1030 FORA=1TO4
1040 POKEFW(A),Q:NEXTA
1050 USR(62)
1060 FORDL=1TO400:NEXTDL
1070 IFQ=74THENQ=P:GOTO1090
1080 Q=74
1090 F=F+1:GOTO1020
1100 CURSOR0,22:PRINT[1,7]"ANOTHER GAME ";NA$(1);" & ";NA$(2);" ? "
1110 PRINT[1,7]" Y/N "
1120 GETSS$:IFSS$=""THEN1120
1130 IFSS$="Y"THEN1160
1140 IFSS$="N"THEN1180
1150 GOTO1120
1160 IFN=1THENN=2:RUN200
1170 N=1:RUN200
1180 PRINT"■THANKS ";NA$(1);" & ";NA$(2):MUSIC"C3D3F#3":END
1190 PRINT"■"
1200 CURSOR10,4:PRINT[5, ]"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
1210 CURSOR10,6:PRINT[5, ]"C O N N E C T   4"
1220 CURSOR10,8:PRINT[5, ]"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
1230 FORDL=1TO300:NEXT
1240 PRINT"■■■FOR 2 PLAYERS"
1250 PRINT"■THE FIRST TO GET 4 TOKENS IN ANY ROW"
1260 PRINT"■IS THE WINNER"
1270 PRINT"■EITHER HORIZONTAL, VERTICAL OR DIAGONAL"
1280 FORDL=1TO2000:NEXTDL
1290 PRINT[1,7]TAB(8); "■ press 'CR' to play "
1300 GETQ$:IFQ$=""THEN1300
1310 IFASC(Q$)=13 THENRETURN
1320 GOTO1300
```

3. Basic Additions

In this chapter there are several additional routines which can be added to S-Basic and saved onto a back-up copy tape. If you just wish to copy the Basic tape in it's original form without any additions, enter the routine below.

3.1 S-BASIC TAPE COPY

It is recommended that your original Basic tape be copied using the following procedure, and keep the original as a Master tape in a safe place away from magnetic fields and to use the copy for normal use, it could be quite expensive to replace the original. This routine requires that the 700 has just been turned on and that S-Basic has not yet been loaded. The display should be showing:-

```
**  MONITOR 1Z-013A  **
*
```

With the cursor next to the asterisk enter:-

MB000 and 'CR'

The display will now appear:-

```
B000 FF
```

With the cursor next to the last F. Enter the values shown under the TO ENTER column followed by the carriage return key after each.

DISPLAYED	TO ENTER
B000 FF	CD
B001 FF	27
B002 FF	00
B003 FF	CD
B004 FF	2A
B005 FF	00
B006 FF	C3
B007 FF	08
B008 FF	11
B009 FF	SHIFT/BREAK

continue at top of next column

```
*
```

The cursor should now be flashing next to the asterisk. Place the S-Basic tape in the tape player and enter:-

JB000 and 'CR' whereupon you will be prompted to press the 'PLAY' button. When Basic has completed loading you will be prompted to 'HIT ANY KEY', but don't do it until you have placed a blank tape in the recorder. Then and only then press any key and the screen prompt will be 'RECORD. PLAY'. S-Basic will now be copied onto the new tape.

It will take six and a half minutes to save the back-up tape, this is because the MZ-700 records programs twice to be on the safe side. If the tape you use finishes before the 'HIT ANY KEY' message is screened again, provided it has been recording for at least three and a half minutes, you can use the 'SHIFT/BREAK' keys to stop the recording and you should still have the back-up copy on the tape, although it would be better to use a C15 cassette and let the saving routine run it's course. When the message is screened again and if you only need one copy press the reset button on the rear panel which is the only way, apart from turning off, to break out of this routine. You can then enter 'J7D79' which will jump into Basic. Copying in this manner can only be carried out by modifying memory before Basic is loaded.

3.2 ADDITIONS TO S-BASIC

To alter the values contained in certain parts of RAM we shall use the RAM Monitor 'Memory set command' described on page 100 of the Owners Manual. When altering parts of memory it is good practise to check that one has entered the correct values by using the RAM Monitor Dump command, which will display the memory contents on screen for checking. First make sure that you are in RAM Monitor, enter 'BYE' if not, then entering 'D219B' and 'CR' will display 128 bytes starting with 219B, whilst entering 'D219B 21A2' will just display a single line of eight bytes. If your 700 has the Printer/Plotter connected entering 'PLOTON' before 'BYE' is entered will also dump the contents to the printer simultaneously. One can enter 'P' when already in RAM Monitor, but the printout is much slower, and it is not displayed on screen, furthermore to abort the Dump, SHIFT and BREAK must be held down longer, until the print head has completed the current line of printing.

In order to make the additional routines permanent on a back-up copy

Basic tape it will be necessary to use a different copying routine from that on the previous page. It is detailed in section 3.2.8

3.2.1 Ready message

This is a fairly simple alteration to make. The message 'Ready' is contained in memory from 219B to 21A0 Hex (8603 to 8608 dec), that is six bytes in total, five for the actual word Ready and the last byte is 00 to define the end of the word. To make a positive check on this is simple.

Enter 'BYE' and 'CR' the cursor will be flashing next to the asterisk. Enter 'D219B' and 'CR' and the 128 bytes starting at 219B will be dumped (displayed) on the screen. The Hex values of Ready will be seen on the first line, to the right of the line are the ASCII repesentations of those values, you will also see the 'Break' message starting at 21A1.

To alter Ready one must first look up in the ASCII code table on page 154 of the Owners Manual to find the Hex values of the letters we wish to change to, do not mistake the ASCII code table for the Display code table on the opposite page they are quite different. If we assume that we would like 'OK GR' displayed instead of 'Ready' the first letter would be O. Find O in the table, it is in column 4 which is the MSD (Most Significant Digit), and line F the LSD (Lowest Significant Digit), therefore it's value is 4F Hex. The letter K is 4B and space is 20. G and R are 47 and 52 respectively.

If you have already entered 'BYE' to place the 700 into RAM Monitor good, if not now is the time to enter it.

With the cursor flashing next to the asterisk enter 'M219B' and 'CR' and the display should appear:-

:219B=52 with the cursor over the 5

Now enter without spaces:-

4F4B204752 and 'CR'

The display will now show:-

:21A0=00

Now enter 'SHIFT/BREAK' and the cursor will be on the next line with the Asterisk. Enter 'R' and 'CR' and you will be back in Basic with the Ready message altered to OK GR, or whatever it was altered to.

3.2.2 Trace command

It is extremely useful to be able to display the current line number of the program being executed when de-bugging programs. As mentioned earlier the two commands TRON and TROFF exist in Basic V1.0A but their operation could be improved.
The next routine alters Basic so that the current line number is displayed in the top left corner of the screen by entering 'TRON' before RUNning the program, whilst 'TROFF' will cancel and return to normal.
As before only enter the characters listed under the TO ENTER column and 'CR' at the end of each series of numbers. Under COMMENTS the Assembly Language Mnemonics are listed for those who understand Z80 code. If you are not into Assembly Language yet, don't worry as they are listed for explanation only.

DISPLAYED	TO ENTER	COMMENTS
Ready	BYE	
*	M2289	
:2289=3E	2AE638	(LD HL,(38E6)
:228C=D3	CDA721	(CALL 21A7
:228F=E6	2100D0	(LD HL,D000
:2292=A7	0605	(LD B,5
:2294=CD	1A	(LD A,(DE)
:2295=51	CDCD04	(CALL 04CD
:2298=5D	CDF200	(CALL 00F2
:229B=17	23	(INC HL
:229C=E1	13	(INC DE
:229D=F1	10F5	(DJNZ 2294
:229F=3A	E1	(POP HL
:22A0=01	F1	(POP AF
:22A1=39	C9	(RET
:22A2=B7	SHIFT/BREAK together	
*	R	(returns to Basic

Holding down 'BREAK' will halt the program, very useful with TRON.

3.2.3 Slow Running

This routine will slow the running of a program when keys 1 to 9 are held down, 9 being the slowest speed. It is especially helpful when using TRON to debug a program. The routine is called continually during program execution, which slows the program a little, therefore one should be able to turn it on only when needed, as when turned off programs will run at normal speed.

POKE6449,47,36 will turn on.
POKE6449,30,0 restores to normal.
One word of warning is that this routine is not too fond of programs with the Basic GET statement, due to the fact they use the same routine in RAM Monitor for scanning the keyboard, therefore if your program contains GET enter POKE89,240 which gives GET repeat on keys held down.
POKE89,83 restores to normal.

DISPLAYED	TO ENTER	COMMENTS
Ready	BYE	
*	M242F	
:242F=CD	AF	(XOR A
:2430=3C	CD5800	(CALL 0058
:2433=07	FE31	(CP 31
:2435=07	3811	(JR C,2448
:2437=CD	FE3A	(CP 3A
:2439=24	300D	(JR NC,2448
:243B=C9	D630	(SUB 30
:243D=13	87	(ADD A,A
:243E=D6	87	(ADD A,A
:243F=30	87	(ADD A,A
:2440=38	47	(LD B,A
:2441=03	0E00	(LD C,0
:2443=0A	0B	(DEC BC
:2444=D8	78	(LD A,B
:2445=78	B1	(OR C

:2446=C9	20FB	(JR NZ,2443
:2448=FE	AF	(XOR A
:2449=2C	CD1E00	(CALL 001E
:244C=1A	C9	(RET
:244D=FE	SHIFT/BREAK together	
*	R	(return to basic

3.2.4 Halt List

During LISTing to the screen holding down the 'SPACE' bar will halt the display until it is released.
This next routine will allow one to halt the list by pressing the 'SPACE' bar, but it will remain halted allowing one to have both hands free to make notes etc.. Pressing the 'SPACE' bar again will single step, line by line, through the listout each time it is pressed thus enabling closer scrutiny of the program, whilst pressing any other key will continue listing as normal.

DISPLAYED	TO ENTER	COMMENTS
Ready	BYE	
*	M4192	
:4192=58	4D24	(address of extra routine
:4194=FE	CC5302	(CALL Z,0253
:4197=F8	00	(NOP
:4198=FE	SHIFT/BREAK together	
*	M244D	
:244D=FE	CD1B00	(CALL 001B
:2450=03	FE20	(CP 20
:2452=FE	C9	(RET
:2453=2F	SHIFT/BREAK together	
*	R	(return to basic

To check on it's effectivness load a long program and list. I think it will prove itself useful.

3.2.5 Bell on Ready

Some users prefer the addition of an audible prompt when the 700 returns to Ready, such as the bell heard when one enters USR(62). It is especially useful if one is loading a lengthy program as one does not need to continually watch the screen for the Ready message, as with loading the Basic tape for instance.

DISPLAYED	TO ENTER	COMMENTS
Ready	BYE	
*	M1870	
:1870=06	8130	
:1872=AF	SHIFT/BREAK together	
*	M3081	
:3081=00	AF	(XOR A
:3082=00	CD3E00	(CALL 003E
:3085=00	CD0600	(CALL 0006
:3088=00	C9	(RET
:3089=00	SHIFT/BREAK together	
*	R	(returns to basic

and you should have heard a beep when Ready was displayed.

3.2.6 Buzz on Error

Whilst on the subject of audible prompts, this routine produces a note whenever Syntax error is encountered.

DISPLAYED	TO ENTER	COMMENTS
Ready	BYE	
*	M2171	
:2171=51	8930	
:2173=2A	SHIFT/BREAK together	
*	M3089	
:3089=00	CD5100	(CALL 0051

```
:308C=00     E5                        (PUSH HL
:308D=00     119530                    (LD DE,3095
:3090=00     CD3000                    (CALL 0030
:3093=00     E1                        (POP HL
:3094=00     C9                        (RET
:3095=00     2D42320D                  (-B2 CR
:3099=00     SHIFT/BREAK together
*            R                         (return to basic
```

To check on the new Error tone after returning to Basic with the Ready message displayed enter a series of keys, such as 'GGGG' and 'CR', the 'Syntax error' will be screened followed by the new tone.

We have entered a Music string into memory, 3095 to 3098 hex, as shown above. If you are not sure of the MUSIC statement check on page 65 of the Owners Manual. The value of 3095 is now 2D hex, which if you check with the ASCII code table on page 154 of the Manual, is the '-' character, which defines the note to be played to be in the lower octave range. Byte 3096 contains the Hex value of the note to played, in this case 42 hex which is note 'B'. And byte 3097 is 32 hex, which defines the duration of the note, duration 2. The string is terminated, as all Music strings must be, with code 0D hex which is a Carriage Return.

Once you have entered this routine it is very simple to alter the note played. If you wish the duration to be longer then altering byte 3097 to 35 will extend the duration from 32 hex (duration 2) to 35 hex (duration 5). Similarly altering byte 3095 from 2D hex (- sign) to 2B hex (+ sign) will play the note in the higher octave. And changing byte 3096 to anything in the range 41 to 47 hex (A to G) will produce that particular note.

Altering a single byte is exactly the same as before, enter 'BYE' and 'CR', and then 'M' followed by the byte to change. The byte number will be displayed along with the current value it contains, overwriting this with the new value followed by 'CR' and the SHIFT/BREAK will complete the task. Then go back to Basic by entering 'R' and 'CR', then test out the new note by entering an incorrect command such as 'GGGG' and the new note will be heard when 'Syntax error' is displayed.

3.2.7 Data tape prompt messages

Some programs use a separate tape to record data, such as the Mail List program in a later chapter of this book. Consequently when the program requires a data tape to be loaded and read through the ROPEN command there could be a 'PLAY' prompt displayed, as when loading a program tape, but there isn't. Quite simply when ROPEN is encountered during program execution the display remains the same, and the unaware user could imagine that the program has crashed or entered an endless loop. The situation is the same when data is to be saved on tape, one could be greeted with the 'RECORD.PLAY' message, but alas it is not displayed.

There is a very simple alteration to Basic which overcomes the problem. Enter:-

```
POKE$4742,$CD:POKE$474C,$AF
```

Altering those bytes will now display both prompts when the commands ROPEN and WOPEN are encountered. Their original values were $CC and $B5 respectively.

Another modification which I feel is required, if using several data tapes, is that the BASIC V1.0A never displays the fact that it has found or is loading a Data tape. With the inclusion of the modification above it now shows 'Found "Data Tape Name"' but doesn't display whether it is loading it or not. The only tell-tale sign that data is in fact loading is that the tape stops momentarily, when the buffer fills up, and then continues, even then it's not that easy to spot. The next routine will display the name of the data tape loading.

DISPLAYED	TO ENTER	COMMENTS
Ready	BYE	
*	M46F7	
:46F7=3A	CD9930	
:46FA=32	SHIFT/BREAK together	
*	M3099	
:3099=00	110B43	(LD DE,430B

```
:309C=00     21FC0F                (LD HL,0FFC
:309F=00     CDC816                (CALL 16C8
:30A2=00     3AFC0F                (LD A,(0FFC)
:30A5=00     C9                    (RET
:30A6=00     SHIFT/BREAK together
*            R                     (returns to basic
```

If you run programs that use data tapes these mods will prove quite valuable.

Any or all of these routines can be added to Basic and then saved onto your back-up copy tape by using the routine in section 3.2.8. Remember that if you have entered any incorrect values in any of the routines, no permanent damage will be done. If you find they do not work correctly probably due to incorrect input, or you get into a muddle, or the program you are running crashes out or becomes stuck in an endless loop, the first thing to try is pushing the reset button on the rear panel which will take you into ROM Monitor followed by SHIFT and 3 keys which will bring you back to Basic, and try running again. If the fault is still there simply switch off for a couple of minutes (never switch off and on quickly) and reload the original (unmodified) Basic tape, and make the alterations again, also checking them using the Dump command to ensure they are correct, and then make the back-up tape.
Here is an example of a Dump of the TRACE function after it has been entered in section 3.2.2. Enter 'BYE' and 'CR' to enter RAM Monitor then 'D2289 22A2' and 'CR'

```
:2289=2A E6 38 CD A7 21 21 00
:2291=D0 06 05 1A CD CD 04 CD
:2299=F2 00 23 13 10 F5 E1 F1
:22A1=C9 B7 C2 CE 20 23 CD 50
```

Checking these bytes with those which should have been entered will confirm if they were entered correctly.

3.2.8 Copying S-Basic with additions

After the 700 loads in the Basic Interpreter 1Z-013B, it is moved down in memory to start at 0000 Hex and the initial screen message containing the copyright notice is deleted from memory, along with the Boot routine which transfers it down to address 0000. Therefore one must re-write the Boot section, and it is a good idea, although not essential, to re-write some form of initial message, as if one has more than one copy it will be clear which one is loaded.

The entries below the COMMENTS are the ASCII codes of the Hex numbers being entered, this first section is re-writing the initial screen message. It begins with code 0D Hex (13 decimal) which is a carriage return, which means after the initial message is printed on screen the cursor will move to the next line. There are two more CR's at the end which will skip two lines before printing 'Ready' after the tape has loaded, the sp represents a space, 20 Hex (32 decimal), between words. Finally all screen messages must end with a NOP, no operand 00 Hex, which determines the end of each message.
MAKE SURE THAT A BLANK TAPE IS IN THE CASSETTE DRIVE AND NOT THE ORIGINAL BASIC TAPE

DISPLAYED	TO ENTER	COMMENTS
Ready	BYE	
*	M6AC1	(position of initial message
:6AC1=C4	0D424153494320	(CR B A S I C sp
:6AC8=00	434F505920	(C O P Y sp
:6ACD=00	315A2D3031334220	(1 Z - 0 3 B sp
:6AD5=00	56312E30410D0D00	(V 1 . 0 A CR CR NOP
:6ADD=00	SHIFT/BREAK together	

This section re-writes the BOOT routine.

*	M6B79	(start of boot routine
:6B79=00	AF	(XOR A
:6B7A=00	110000	(LD DE,0000

```
:6B7D=00    CD3300                 (CALL 0033
:6B80=00    2A7111                 (LD HL,(1171)
:6B83=00    225412                 (LD (1254),HL
:6B86=00    110000                 (LD DE,0000
:6B89=00    21796B                 (LD HL,6B79
:6B8C=00    B7                     (OR A
:6B8D=00    ED52                   (SBC HL,DE
:6B8F=00    4D                     (LD C,L
:6B90=00    44                     (LD B,H
:6B91=00    210012                 (LD HL,1200
:6B94=00    D3E0                   (OUT (E0),A
:6B96=00    EDB0                   (LDIR
:6B98=00    C30000                 (JP 0000
:6B9B=00    SHIFT/BREAK together
```

There are now some calls within the Interpreter which must be altered, afterwhich is the section to move Basic up to 1200 Hex, which is where it will load from.

```
*           ME000                  (routine to move up interpreter
:E000=00    212218                 (LD HL,1822
:E003=00    22E700                 (LD (00E7),HL
:E006=00    3E00                   (LD A,00
:E008=00    32CF6B                 (LD (6BCF),A
:E00B=00    32D16B                 (LD (6BD1),A
:E00E=00    21A06B                 (LD HL,6BA0
:E011=00    11A07D                 (LD DE,7DA0
:E014=00    01A16B                 (LD BC,6BA1
:E017=00    EDB8                   (LDDR
:E019=00    C36A4C                 (JP 4C6A
:E01C=00    SHIFT/BREAK together
```

Now for the actual copying

```
*           GE000                  (GOTO move up routine
```

The screen will now clear and display:-

```
**  MONITOR 1Z-013A  **            (The original ROM Monitor
                                   (message is displayed
*            S12007DA07D79          (ROM Monitor Save Command
FILENAME?    BASIC COPY             (Enter any name up to 16 letters
⤓ RECORD.PLAY                       (Press the two cassette keys
WRITING  BASIC COPY
OK'
```

Please note that whenever using the SAVE command in <u>ROM</u> Monitor, as you have just done and is described on page 149 of the Owners Manual, there should NOT be any spaces between the save parameters. As in the above case we have saved the program from address 1200 Hex to 7DA0 to execute at 7D79. If we were saving a machine code program using the RAM Monitor there would be spaces between them, as shown on page 100 of the Manual.

Once recording is completed you will still be in ROM Monitor, and to re-enter Basic one must enter J7D79.

Once again it will take six and a half minutes to save the back-up tape. If the tape you use finishes before the 'OK' message is screened, provided it has been recording for at least three and a half minutes, you can use the 'SHIFT/BREAK' keys to stop the recording and you should still have the back-up copy on the tape.
Any further copies of this modified Basic must be made using this section and not the routine in section 3.1.

4. PEEKS & POKES

This chapter deals with areas of S-BASIC which can be PEEKed to find information stored, or POKEs and USR calls which can be made for different effects.

It is important to remember that when PEEKing a location, to find what value it contains, whether one enters the byte in hexadecimal i.e. PRINT PEEK($6BCF), or decimal i.e. PRINT PEEK(27599) both these statements refer to the same location, the decimal value the byte contains will be displayed and NOT the hex value.
The leftmost numerals are in decimal followed by the Hex equivalent.

21....$0015 Entering USR(21,Z$) prints a message contained in string Z$ on screen

48....$0030 Entering USR(48,M$) will play music contents of string M$. i.e.M$="+C+DE2"

62....$003E Entering USR(62) will sound a short note, ideal for giving an audible prompt whenever input is required.

68....$0044 USR(68) turns on tone generator.

71....$0047 USR(71) turns off tone. See chapter 2.

77....$004D PLOTON flag, normally contains 0.
POKE77,1 will direct all printing to the printer and screen.

78....$004E CONSOLE flag, normally contains 0. It contains 1 after CONSOLE parameters have been entered. i.e. CONSOLE2,15,5,30.

84....$0054 Contains current position of cursor column. The columns are from 0 (left) to 39.
POKE84,6 will move the cursor to column 6, similar to TAB(6).

85....$0055 Contains current position of cursor line. The positions of the cursor lines are from 0 (top line) to 24 (bottom line).
POKE85,20 will move the cursor to line 20 irrespective of it's previous line. This could be simpler than using the CURSOR0,20 statement as only one parameter is needed.

86....$0056 If CONSOLE has been entered with parameters i.e. CONSOLE 2,15,5,30 it will contain the starting line of the scroll area, in this case 2.

87....$0057 Contains the number of the finish line of the scroll area, and NOT the amount of lines to print, in the example shown it is 16 (first line is 0 remember) so 16 will represent line 17 (2+15)

89....$0059 There is no auto repeat on GET. Usually it is not needed except for continuous input in games programs where it is important to keep down a particular key, rather than keep taking the finger off and re-pressing it.
POKE89,240 will allow constant input from one key.
POKE89,83 reverts to normal.

91....$005B Contains starting column of scroll area, in the example it is 5

92....$005C Contains number of the column to finish at in the scroll area, in this case column 34, column 1 being 0 (5+30=35)

93....$005D Contains the value of current colour co-ordinates of the screen. After loading S-BASIC, with the screen printing white characters on a blue background, it will contain 113. If one converts this decimal number to hex it will be seen to be 71hex. The first number (7) relates to the character colour (7 is white) and the second the background colour (1 is blue).
POKE93,$15 will alter the colours to Blue on Light Blue.
POKE93,$27 is Red on White.
POKE93,$A7 will print the next statement in the second Display character set in Red on White.

Example:-
POKE93,$A7:PRINT"SHARP RULES"
will print out the message in the outline characters of the second Display code table, all further printing reverts to normal.
The values of the colours are on page 80 of the Owners Manual.

95....$005F Contains the ASCII value of the last key pressed.

96....$0060 Contains the Display code value of the current cursor. When in Alpha mode the cursor character is 239 (EF hex), in Graph mode it is 255 (FF hex) and 67 (43 hex) when SHIFT/ALPHA have been pressed for lowercase (small) letters mode.
The Alpha (normal) cursor can easily be altered to any other character.
POKE96,$2B will alter it to the = sign. Note use the Display code table. The cursor will revert to it's previous character if one changes modes i.e. from Alpha to Graph and back again. For a permanent cursor change see $07F9.

100....$0064 AM/PM flag. Will contain 0 if TI$ is under 120000 hours (AM), 1 if more (PM).
POKE100,1 would alter TI$ from 032059 to 152059.

292....$0124 POKE292,1 will sound the bell when 'CR' is pressed.
POKE292,4 reverts to normal.

399....$018F POKE399,150 sounds the bell on all other keys pressed except CTRL, SHIFT, CR
POKE399,153 reverts to normal. This could be useful on inputs required from the keyboard. The bell will sound after one presses the DEF KEYs as each character is displayed.

648....$0288 Contains the speed at which the cursor moves. At present it contains 10.
POKE648,5 will increase it's speed dramatically, whilst POKEing it with a higher value than 10 will slow it down.

1203....$04B3 Holding down the BREAK key during program execution will temporarily halt the program. It may be that you don't want your programs halted.
POKE1203,201 will disable the function.
POKE1203,216 reverts to normal.

1621....$0655 The Routine for the Scrolling up effect of the screen, shown on page 129 of the Owners Manual, whereby holding down the SHIFT and cursor keys scrolls the complete two screenfuls round and round, is stored here. If one wants to make use of the scrolling during program running a USR call to this address is needed. To scroll the screen one complete turn, so the cursor ends up where it started after displaying the whole of Video Ram (2 Kbytes), one could enter this line:-

```
FORX=1TO50:USR(1621):NEXT
```

Try entering it in direct mode (without a line number) to see the effect before using it in a program.

1661....$067D For This routine controls scrolling down:-

```
FORX=1TO50:USR(1661):NEXT
```

Will scroll down.
There appears to be a difference of opinion in what one calls up and down. On page 129 of the Manual it describes holding SHIFT with the Cursor UP key for scrolling up, but my 700 scrolls down. Similarly SHIFT with Cursor DOWN keys scrolls up. I suggest one tries both the above to decide what you call up and down.

2041....$07F9 Contains the alpha cursor seed value 239 (EF hex) in Display code.
POKE2041,79 will alter it to ':'. Any value can be entered, so experiment. After entering the cursor will not change immediately as the current character is stored at $0060, press the Graph key and then the Alpha key and it will change.
POKE2041,239 will revert to normal.

2610....$0A32 Holds Tempo value. If TEMPO is set to 1 this byte contains 7. If set to 6 the byte holds 2 and so on, if TEMPO is set to 7 the byte holds 1. When S-BASIC is first loaded it is set to 4.

2617/8....$0A39/A These two bytes control the tone generator described in chapter 2.

4092....$0FFC Contains details of last program loaded. It will hold 5 if it was a Basic program, 4 if Data tape, 2 if MZ-80K Basic program.

4093-4109....$0FFD-$100D Contains Name of last program loaded.
To find out the name enter:-

```
FORX= $0FFD TO $100D:?CHR$(PEEK(X));:NEXT
```

or in decimal:-

```
FORX= 4093 TO 4109:?CHR$(PEEK(X));:NEXT
```

4110/1....$100E/F Contains size of last program loaded. A single byte can only hold a value up to 255, so in cases where the number is likely to exceed 255 two bytes are required. The first always holds the LSB (least significant byte) and the second the MSB (most significant byte). Therefore to determine the decimal value contained in a pair of bytes one should multiply the second byte by 256, and add the two together:-
PRINTPEEK(4110)+PEEK(4111)*256 will print the size of the last program loaded, remembering that the multiplication sign takes priority over the addition sign and will be carried out first.

4112/3....$1010/1 Holds in two bytes the loading address of the last program loaded.
PRINTPEEK(4112)+PEEK(4113)*256 will reveal 27599 for Basic programs, as this is where they load from ($6BCF).

6190....$182E Contains $FF. On loading Basic clears memory contents up to $FF00. Altering this byte to $A0 would protect any data in memory above $A000 from being wiped out on loading Basic, but it would need to be copied onto a back-up tape.

10003....$2713 The INPUT prompt '?' is stored at this location. POKE10003, with any ASCII code value will change it. POKE10003,63 restores to normal.

Additional sections of the Interpreter are listed in chapter 8.

The ASCII code table below 32 decimal is used as control codes. PRINT CHR$(followed by one of the numbers below will produce these effects:-

CHR$(5) Alters keyboard to lowercase input, the same as pressing SHIFT, therefore keys pressed other than alpha will produce the shifted character. Key 2 will produce ".
CHR$(6) Changes back to uppercase.
CHR$(13) Issues a Carriage return.
CHR$(16) Deletes one character.
CHR$(17) Moves the cursor down one line.
CHR$(18) Cursor up one line.
CHR$(19) Moves the cursor one position right.
CHR$(20) Cursor one position left.
CHR$(21) HOMES cursor.
CHR$(22) Clears screen. i.e.CLS.
CHR$(23) Alters keyboard to Graphic mode.
CHR$(24) Prints a space.
CHR$(25) Alters keyboard to normal Alpha mode.
CHR$(27) Same as 13.

The listing on this page prints a patchwork of patterns and colours. It uses the full 512 characters from both Display codes, and is very relaxing to view.

```
10 COLOR,,,0
20 PRINT"■"
30 F=1:B=7
40 W=0
50 FORX=0TO6
60 VR=53252
70 FORY=0TO4:FORZ=0TO4
80 AD=(W*200)+(X*5)+(Y*40)+Z
90 POKEVR+2048+AD,D
100 POKEVR+AD,A:NEXTZ
110 NEXTY
120 D=INT(RND(1)*2)
130 A=INT(RND(1)*255)+1
140 IFD=1THEN D=128+F*16+B:GOTO160
150 D=F*16+B
160 F=F+1:IFF>7THENF=0
170 B=B+1:IFB>7THENB=0
180 NEXTX
190 W=W+1:IFW>4THENGOTO40
200 GOTO50
```

The next short listing will also test the definition on your T.V. It prints symmetric patterns on screen in randomly selected colours. The ASCII codes of the characters are listed in the DATA line 230, alter them to different characters if you wish by checking in the table in the Appendix. One could add additional characters, but make sure line 40 contains the total amount in the DATA statement.

```
10 CONSOLE:COLOR,,7,0
20 PRINT"▣"
30 F=RND(1)*7+1:S=F+2:IFS>7THENS=2
40 FORA=1TO11
50 READD
60 FORB=1TO100
70 A$=CHR$(D)
80 FORC=1TO2
90 FORDL=1TO50:NEXT
100 Y=RND(1)*12+1
110 X=RND(1)*20+1
120 CURSORX,Y:PRINT[F,S]A$
130 CURSORX.24-Y:PRINT[F,S]A$
140 CURSOR40-X,Y:PRINT[F,S]A$
150 CURSOR40-X,24-Y:PRINT[F,S]A$
160 A$="■"
170 NEXTC,B
180 F=F+1:IFF>7THENF=0
190 S=S+1:IFS>7THENS=0
200 NEXTA
210 RESTORE
220 GOTO40
230 DATA167,172,149,112,124,243,241,145,108,200,246
```

5. Plotter Printer

Even with some previous knowledge of Basic there may be some commands in the Colour Plotter-Printer section which one may feel need illustrating.

5.1 Text Mode Commands

The Text mode commands appear fairly straightforward as to the selection of character sizes. Entering MODE followed by either TN, TS or TL will print in 40, 80 or 26 characters per line respectively.

SKIP

Must be followed by either a number or variable, and will feed the paper if it is positive, or back-feed if negative. A variable which is not an integer, i.e.11.76, will be truncated, in this case to 11. PEEK20031 will display the value contained as 21 which is the figure at which 'Illegal data error' will occur if the positive value entered exceeds 20.

POKE20031,41 would allow up to 40 lines to be SKIPped.

PEEK20027 contains 236 which is the negative default value (256-20)

POKE20027,216 will allow up to SKIP-40.

PAGE

Sets the number of lines per page, until a line is left blank which denotes the end of page. The present maximum is 72.

PEEK21068 contains 73, the default value.

POKE21068,93 would allow up to 92 lines per page.

LIST/P

Although similar to listing to the screen the Plotter cannot print graphic characters. These are LISTed displaying their hex ASCII codes, and in a different colour to that being used.

5.2 Graphic Mode Statements

These Graphic mode statements illustrated make reference to the range of the X and Y co-ordinates as points (X=east/west Y=north/south). The maximum width of the plotter paper (X axis) is 480 points, whilst the maximum Y axis is limited by the Basic Interpreter, not by paper length, to 999 points (north) to -999 (south).

LINE and RLINE

These commands differ only in their referal to their origins. LINE must be followed by values which represent the position to draw to in respect of it's starting position in that sequence, or after HSET has been used.
Enter MODEGR and 'CR'
Enter LINE240,0 'CR' and a straight line will be drawn to the centre of the paper.
Now enter LINE360,0 'CR' and the line will continue another 120 points, finishing at point 360 in relation to the start.
RLINE is followed by values which relate to it's position before that particular statement.
Enter PHOME 'CR' then feed the paper on a few lines.
Enter RLINE240,0 'CR' and once again it will draw to position 240.
Now enter RLINE120,0 'CR' which will draw to point 360 across the paper. So the reference was made to it's present position and not the starting position as used with LINE.
Care must be taken when using RLINE, if the print head is already at point 360 across the paper and one enters RLINE480,0 don't be surprised if the print head tries to move into the cassette unit.

MOVE and RMOVE

Both these statements lift the pen and move to a specified point. They both move in relation to separate origins as above. LINE refers to it's start position and RLINE to it's present position.

To illustrate the above statements these next lists both produce the same results, that of a square in the centre of the paper.

```
 Using LINE                                                     Using RLINE
10 MOVE120,0        :X move to 120, Y level:                  10 RMOVE120,0
20 LINE360,0        :X draw from 120 to 360, Y level:  20 RLINE240,0
30 LINE360,-240   :X level, Y draw down to -240:       30 RLINE0,-240
40 LINE120,-240   :X draw from 360 to 120, Y level:  40 RLINE-240,0
50 LINE120,0        :X level, Y draw up 240 points:     50 RLINE0,240
```

Studying these alternatives should explain the differences, and of course either can be entered as one line:-

```
10 RMOVE120,0:RLINE240,0,0,-240,-240,0,0,240
```

To change the line drawn from a solid to a broken line, LINE or RLINE must be suffixed with the % sign and followed by a value between 2 and 16, the types of line are shown on page 86 in the Manual. Remember that after altering the line type all future drawing will be in the last type specified unless redefined.

PHOME

Will return the print head to the original points of X and Y.

HSET

Alters the origin of X and Y to the present position, any subsequent PHOME will return to this point.

GPRINT

The only way to print alpha or numeric characters when in MODEGR is by using this statement. It will print at the current print head location unless MOVE has been entered.

GPRINT A$ will print the contents of string A$

GPRINT "BAR CHART" will print whatever is within "".

The angle of printing can be altered to one of 4, along with the size of the characters up to 63. Apart from it's obvious uses, it could be used to print small banners. The example below will print 'HAPPY BIRTHDAY' in the largest size, so make sure there is enough paper. If not shorten the string to one letter to demonstrate, one could add someones name to it, or alter the string completely.

```
10 MODEGR:MOVE0,0:HSET
20 FORA=0TO3:MOVEA*3,-A*3:PCOLORA:GPRINT[63,1],"HAPPY BIRTHDAY":NEXT
```

AXIS

Draws lines with scale marks at given intervals. i.e. AXIS 1,50,9 The first value to follow AXIS denotes the direction and must be either 0 to denote the Y axis (north/south) or 1 for the X axis (east/west). The second relates to the distance between each scale mark, in the above case 50 points, which is roughly equal to 1 centimetre. The third figure is the number of repititions. Care must be taken with the amount of repeats, remembering the paper is only 480 points wide, so in the above example the print head would finish at point 450 (50*9). As a general rule if one is drawing a scale line across the full width of the paper the second and third parameters multiplied together must not be higher than 480, else the print head may be shaking hands with the cassette again.
i.e. 240,2 160,3 120,4 96,5 80,6 etc.
AXIS1,2,240 will print a minute scale across the whole paper width. The Y axis amount of repititions is governed, not by length of paper, but within the Interpreter from -999 to 999.

CIRCLE

Is followed by six parameters. The first two refer to the X and Y centre of the circle starting point. If the print head is at the left side of the paper, entering the first two figures as 60,0 would make point 60 on the X axis the centre of the circle whilst keeping the Y axis level. Altering the second from 0 to within the range -999 to 999 would move the paper either down or up to the correct Y axis.
The third figure relates to the radius of the circle, and once again is in the range -999 to 999. The first (X axis) and third (radius) parameters must be entered with thought, as we know the paper being 480 points wide, if the centre of the X axis was 360 and a complete circle was to be drawn from there, then the highest figure for the radius could only be 120, otherwise a full circle would not have enough paper (width) to be drawn. The fourth parameter is the starting angle, if 0 it will start from the 3 o'clock position, if 90 the 12 o'clock, and 180 the 9 o'clock etc., in an anti-clockwise manner. The fifth figure is the ending angle, as with the starting angle 0 is equal to 3 o'clock, so if only the top half of a circle

was needed then these parameters would be entered as 0,180, the 180 being at 9 o'clock. The sixth and last value is the step angle of each plot made, obviously the smaller the angle the smoother the circle becomes. In the example shown in the Manual on page 90 a step value of 0.2 is given, this slows down the actual drawing speed dramatically. Try higher settings for the step angle, even 10 gives a fairly neat circle, and speeds up the drawing process 20 times.

```
CIRCLE240,0,90,0,360,10
```

Draws a circle with the centre at 240 on the X axis, centre of paper
The Y axis is level at 0
The radius is 90 points, making the total diameter 180 points
The starting angle is 0, at 3 o'clock
The finishing angle is 360, a complete circle
And the step angle is 10

The previous examples assume the print head to be at the left position. If it isn't press the reset switch on the printer with a pencil or similar pointed object, and enter MODEGR.

PRINTER TIPS

If not in constant use the printer pens may dry-up, therefore when not in use remove from the 700 and remember to keep them capped.
Do not use paper rolls less than 110 mm wide. If one looks carefully, small pin pricks are made on the left and right of the paper when in MODEGR to assist correct feeding.
Do not operate with the Printer cover removed, as it is too easy do drop something into the printer.

The following program on Biorhythms uses the plotter to print out the charts, it can still be run if you have not got the plotter fitted, delete lines 880 to 1400 inclusive.

```
10 CONSOLE:COLOR,,7,1
20 PRINT"▣";:PRINT[1,7]TAB(10)" B I O R H Y T H M "
30 DIMX(11),M$(12),D(12):X$=" 0123456789":A=53248
40 FORI=1TO11:READX(I):NEXTI:FORI=1TO12:READM$(I),D(I):NEXTI
50 PRINT
60 INPUT"Enter Name.............";N$
70 INPUT"Year born (i.e.1969)...";YB
80 IFYB<1901 THEN PRINT "SORRY CANNOT DO":GOTO60
90 INPUT"Month     (1 to 12)....";MB
100 IF(MB<1)+(MB>12)THEN90
110 INPUT"Day       (1 to 31)....";DB
120 IF(DB<1)+(DB>31)THEN110
130 PRINT:PRINT
140 INPUT"Year to start..........";YS
150 IFYB<1901 THEN PRINT "SORRY CANNOT DO":GOTO140
160 INPUT"Month..................";MS
170 IF(MS<1)+(MS>12)THEN150
180 DS=1
190 INPUT"How many Months........";HM
200 Y=YB:M=MB:D=DB:GOSUB650
210 GOSUB690:G=F:Y1=YB:M1=MB:D1=DB
220 Y=YS:M=MS:D=DS:GOSUB650
230 GOSUB690:H=F
240 IF(M<3)*(Y/4=INT(Y/4))*(Y1/4<>INT(Y1/4))THENG=G+1
250 IF(M>2)*(Y/4=INT(Y/4))*(Y1/4=INT(Y1/4))THENG=G-1
260 IF(Y/4<>INT(Y/4))*(Y1/4=INT(Y1/4))THENG=G-1
270 N=G-H:IFN>0THENGOSUB680:GOTO220
280 P=23*(N/23-INT(N/23)):Q=28*(N/28-INT(N/28))
290 R=33*(N/33-INT(N/33))
300 L=32-LEN(M$(M1)+STR$(D1)+STR$(Y1))
310 PRINT"▣Name: ";N$;TAB(L);"BORN: ";M$(M1);D1;Y1
320 COLOR,,0,5
330 HM=HM-1
340 PRINT"▣ ●=Physical 0=Emotional □=Interllectual"
350 PRINT"    ┌┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┐ ":FORI=0TO7
360 PRINT"    │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ ":NEXTI
370 PRINT"    ├─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┤ ":FORI=0TO7
380 PRINT"    │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ ":NEXTI
390 PRINT"    └┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┘ "
400 NR=-1:S=Y:T=M:U=D:VV=0:W=21
410 FORI=0TO35
420 IF(U=1)*(NR=-1)THENNR=I
```

```
430 C$=RIGHT$("  "+STR$(U),2):IFI/2<>INT(I/2)THEN480
440 B$=LEFT$(C$,1):GOSUB760
450 POKEA+3+(W*40)+VV+I,B
460 B$=RIGHT$(C$,1):GOSUB760
470 POKEA+3+(W+1)*40+VV+I,B
480 U=U+1:IFU<=D(T)THEN510
490 IF(T=2)*(S/4=INT(S/4))*(U=29)THEN510
500 U=1:T=T+1:IFT>12THENT=1:S=S+1
510 NEXTI:IFNR>30THENNR=30
520 COLOR,,7,1
530 PRINT"▓▓    ";M$(M);",";STR$(Y);
540 NM=M+1:IFNM=13THENNM=1
550 PRINTTAB(35);M$(NM)
560 FORI=0TO34
570 K=INT(8*SIN((-P+I)*2*π/23)+.5):B=71:GOSUB800
580 K=INT(8*SIN((-Q+I)*2*π/28)+.5):B=72:GOSUB800
590 K=INT(8*SIN((-R+I)*2*π/33)+.5):B=74:GOSUB800
600 NEXTI
610 PRINT[1,7];TAB(14);"▓ HARD COPY Y/N ":USR(62)
620 GETA$:IFA$=""THEN620
630 IFA$="Y"THEN880
640 GOTO1410
650 E=1:IF(D<=D(M))THENRETURN
660 IFM<>2THEN680
670 IF(Y/4=INT(Y/4))*(D=29)THENRETURN
680 E=0:USR(62):PRINT"▣INCORRECT DATE":MUSIC"R9":GOTO70
690 F=0:IFM<2THEN740
700 FORI=1TOM-1
710 F=F+D(I)
720 NEXTI
730 IFY/4=INT(Y/4)THENF=F+1
740 F=F+365*Y+INT(Y/4)+D
750 RETURN
760 C=1
770 IFB$=MID$(X$,C,1)THENB=X(C):RETURN
780 C=C+1:IFC>11THENB=0:RETURN
790 GOTO770
800 AD=A+3+(11-K)*40+I
810 POKEAD,B:RETURN
820 DATA0,32,33,34,35,36,37,38,39,40,41
830 DATAJAN,31,FEB,28,MAR,31,APR,30,MAY,31,JUN,30
840 DATAJUL,31,AUG,31,SEP,30,OCT,31,NOV,30,DEC,31
```

```
850 M=M+1:IFM>12THENM=1:Y=Y+1
860 D=1
870 GOSUB690:H=F:GOTO270
880 MODETN:PCOLOR0:PRINT/P;"Name ";N$;TAB(L);"Born: ";M$(M1);D1;Y1
890 PCOLOR0:PRINT/PTAB(16);"High"
900 MODEGR
910 LINE%1,442,0,442,-240,0,-240,0,0
920 FORX=2TO32STEP4
930 MOVEX*13,0
940 LINE%7,X*13,-240
950 MOVE(X+2)*13,-240
960 LINE(X+2)*13,0
970 NEXT
980 MOVE0,-120
990 HSET
1000 LINE%1,442,0
1010 Z=P:X=23:PC=1:GOSUB1080
1020 RMOVE4,0:GPRINT[0,0],"PHYS."
1030 Z=Q:X=28:PC=2:GOSUB1080
1040 RMOVE4,0:GPRINT[0,0],"EMOT."
1050 Z=R:X=33:PC=3:GOSUB1080
1060 RMOVE4,0:GPRINT[0,0],"INTEL."
1070 GOTO1170
1080 PHOME
1090 K=INT(120*SIN((-Z+I)*2*π/X)+.5):MOVE0,K
1100 PCOLORPC
1110 FORI=0TO34
1120 K=INT(120*SIN((-Z+I)*2*π/X)+.5):GOSUB1150
1130 NEXTI
1140 RETURN
1150 IFI=0THENMOVE0,K:RETURN
1160 LINE13*I,K:RETURN
1170 PHOME
1180 MOVE0,-160
1190 HSET
1200 PCOLOR0
1210 MODETN:PRINT/PTAB(17);"Low":MODEGR
1220 MODEGR
1230 NR=-1:S=Y:T=M:U=D:VV=0:W=21
1240 FORI=0TO35
1250 IF(U=1)*(NR=-1)THENNR=I
1260 C$=STR$(U):IFI/2<>INT(I/2)THEN1310
```

```
1270 GPRINT[1,1],C$
1280 IFLEN(C$)>1THENZ=24:GOTO1300
1290 Z=12
1300 RMOVE26,Z
1310 U=U+1:IFU<=D(T)THEN1340
1320 IF(T=2)*(S/4=INT(S/4))*(U=29)THEN1340
1330 U=1:T=T+1:IFT>12THENT=1:S=S+1
1340 NEXTI:IFNR>30THENNR=30
1350 MODETN
1360 SKIP2
1370 PRINT/P;M$(M);",";STR$(Y);
1380 NM=M+1:IFNM=13THENNM=1
1390 PRINT/PTAB(33);M$(NM)
1400 SKIP4
1410 IFHM>0THEN850
1420 END
```

6. Program Tips

6.1 Recovering Lost Programs

If one has ever typed NEW by mistake and thought they had lost a program this section will endeavour to show you how to bring it back to life. After NEW has been entered all variables are zeroed, just the same as entering CLR, and the pointer which holds the position in memory of the end of the file (program) is altered to the position next to the start of file, the beginning of where programs start in memory. Your program, or 99% of it, is still in memory only the 700 doesn't think it is. In addition the length of the first line, which is the very first byte in the program storage area is also set to zero, hence if one enters NEW then LIST nothing gets listed as it doesn't think there is a program there. So there are a few bytes which will have to be altered in order let it believe that it still contains a program. To understand it a little more it would be better if you have just loaded Basic afresh and the memory does not contain a program.

When one enters lines in a program they are converted in a way that the computer understands, for instance if a line contained INPUT A$ the keyword INPUT would not be stored in memory as a word but a unique hex number, a token, which in this instance would be 91 hex. Enter the next 2 line program and then we will dump the appropriate section of memory to see how a program is stored.

```
10 INPUTA$
20 PRINTA$
```

Enter 'BYE' and 'CR' to jump into RAM Monitor.

Now enter 'D6BCF' and 'CR' and the contents of the 128 bytes starting at 6BCF hex will be displayed. 6BCF hex is where all Basic programs commence in memory, and this first byte contains the length of the first line in the program, in this case 8. The third and fourth bytes are the actual line number one has given to the first line. A single byte can only hold a number up to FF hex (255 dec), so whenever it may be neccessary to hold a larger number 2 bytes are used and the lower byte comes first. Basic line numbers can be as high as 65535 and no higher, but in our case the line number was 10 which is stored as 0A hex. The fifth byte contains the token for the keyword INPUT, 91 hex, this is followed by the ASCII codes for A and $, and the last byte of a line is always 0. The second line is very similar with the exceptions of the line number, which was 20 and is stored as 14 hex, and the token for PRINT is 8F hex.

```
        line         line      token   A     $
       length         no.      INPUT
:6BCF=08     00    0A     00    91     41    24    00
```

```
        line         line      token   A     $
       length         no.      PRINT
:6BD7=08     00    14     00    8F     41    24    00
```

If one enters 'R' and 'CR' to return to Basic and then enters 'NEW' the program will not LIST or RUN, because the pointers have been zeroed.
Enter 'BYE' and 'CR' again followed by 'D6BCF'. One can see that only the first line has been changed, the second line is still intact. They should look like this

```
:6BCF=00     00    00     00    91     41    24    00
:6BD7=08     00    14     00    8F     41    24    00
```

The first task is to calculate the length of the first line, in this instance we know it was 8, but normally a program line would take up more bytes than that and one should look for the start of the second line, and count how many bytes there are between the start, 6BCF, and the first byte of line 2. Altering the bytes contents is very simple, move the cursor, by the cursor control keys, up and across so that the cursor is over the first digit of the first byte and enter 08 followed by 'CR'. This has now altered the contents of the byte, but remember that one must always use a hex number so if it

was found that the first line was 26 bytes long one should convert this to hex before overwriting the byte, which would be 1A in that case. The line still isn't the same as before we entered NEW, the actual line number is still zero (byte 3), but in fact it would RUN with a first line as 0. When entering Basic lines one cannot enter a line number as 0, but nevertheless at the moment it is zero and will LIST and RUN.
Before going back to Basic and testing the program there is still the end of file pointer to be altered. Move the cursor back down to the asterisk and one position to the right of it and enter 'D6AB3' and 'CR'. The only pair of bytes we are interested in here is the first two which have D1 6B as there contents. This is, in reverse order, the address of the end of file 6BD1, which must be changed. Usually it contains the address of two bytes after the end of the current program. Normally a program would be much longer than this example and it would help if one could remember some of the contents of the last line. If one dumps from 6BCF onwards, by continually entering 'D' after each section has been displayed, one might be able to see some of the text to the right of the bytes contents. Unfortunately keywords, as they are tokenised, won't be too obvious to spot but print strings will be. Once the end has been found the address must be entered into 6AB3, but don't worry too much if you aren't exactly sure of the last byte of the program. It does not have to be exactly two bytes after the end, 1000 bytes after would do providing the program isn't too long, just make sure it is after the end. Enter 'D6AB3' and 'CR' again. If it was decided that the program ended at 7BD0 for example, we must enter 7BD2 or more into bytes 6AB3 and 6AB4. The last byte is entered first in 6AB3 so position the cursor as before over 6AB3 and enter 'D2', then move it to the right once so that now it is over the next byte and enter '7B' and 'CR'. Moving the cursor back down and adjacent to the asterisk one can enter 'R' and 'CR' to go back to Basic and the program should LIST. Check if the last line is as you had remembered it and if not alter it, but the main bulk of the program should be displayed, then try running it.

6.2 Converting MZ-80K to MZ-700

S-Basic contains an automatic conversion routine to alter Basic programs on tape written for the Sharp MZ-80K to run on the MZ-700. However it will not convert MZ-80A programs. As the MZ-80K had a built in monochrome screen there will not be any color commands in these programs, they will have to be added as required.

After loading the program in the normal way the Basic interpreter will know from reading the data at the start of the program that it is in K format. Whereupon the message 'CONVERTING TEXT' will be displayed followed by a series of dots similar to when MERGEing a program, one for each line of the program changed. Once this is completed the 'Ready' message will be screened BUT before attempting to run one should list the program on screen as there may be further alterations to be made if the program contains any PEEKs or POKEs. Any POKEs to the Video Ram area (53248 to 54247) can remain as these are compatible. All other POKEs must be checked with the list below and altered, and if they aren't there then look for a machine code routine in the listing which is POKEd into the top of memory and uses the LIMIT statement. If this is the case the chances are that the program will not run correctly, but try anyway as the worst that will happen is that the program will crash and you may have to load Basic all over again.

These are the addresses which may be POKEd and must be altered:-

4465/6 decimal controlled the cursor positioning. The K did not have the CURSOR command and a typical line might be:-

```
POKE4465,20:POKE4466,10
```

4465 was the column and 4466 the line. One can alter it to `POKE84,20:POKE85,10` or enter `CURSOR20,10`

The K did not have repeat facilities on any key held down such as when used with a GET statement (neither has the 700, but see chapter 4 for the mod) and it was overcome by PEEK17828 which contained the ASCII code of the last key pressed. One might see a line GETA$ followed by IF PEEK(17828)=81 THEN etc.. It is simply testing for a particular key, in this case the 'Q' key, to see if it is continously being held down. Alter the line to IF A$="Q"THEN etc.

and if one wants to test if it is held down use the mod in this chapter.

The two bytes used with the Tone generator that makes all those funny noises were 4513 and 4514, these should be altered to 2617 and 2618 respectively.

POKE4464,1 altered the keyboard to accept lower case (small) letters, this can be overcome by changing it to PRINTCHR$(5). POKE4464,0 restored to upper case letters and should be changed to PRINTCHR$(6).

Any USR(62) calls should remain as should USR(68) and USR(71).

Any other POKEs should be viewed with scepticism and deleted from the program. There may be POKEs to 10167 which simply turned off a flag which stopped the owner PEEKing the main memory, very frustrating that was, and if one left it in the on the 700 it would wipe out part of the ?USING routine. So be prepared to delete any others, as the addresses most certainly won't be compatible with the 700, and were probably trying to achieve special effects which we take for granted and are included in the Basic of the 700 anyway.

6.3 Logical Operators AND/OR

In many versions of Basic, and also in program listings in Magazines, one may encounter a line such as:-

```
100 IF Z=9 AND Y=8 THEN GOTO 800
```

or:-

```
200 IF Z=2 OR Y=6 THEN GOTO 600
```

Sharp S-Basic does use these operators but not in this form. The above lines should be entered as:-

```
100 IF (Z=9)*(Y=8) THEN GOTO 800
```

or:-

```
200 IF (Z=2)+(Y=6) THEN GOTO 600
```

The * sign should be substituted for AND

The + sign substituted for OR

And both arguments enclosed in brackets.

6.4 User Prompts

One has probably seen in programs, or written into your own programs the 'PRESS ANY KEY' prompt, followed by a GET statement. This can cause confusion to non-computerists as to whether or not they have pressed the correct key, as pressing the 'SHIFT','CTRL' or 'BREAK' keys would have no effect whatsoever with GET. It is far better to specify the key to be pressed such as 'PRESS SPACE' or 'CR' and test for that key in the next line. One cannot enter a line such as IF A$="CR" but one can test for the ASCII value of any key. For instance the ASCII value of the 'CR' key is 13 and the line could be:-

IF ASC(A$)=13 THEN which would only accept 'CR' as being pressed. The codes of all the keys are in the ASCII table with their decimal values in the Appendix.

6.5 Delay Loops

To display a message on screen for a limited time before continuing you may have used a delay FOR..NEXT loop:-

```
100 FOR DL=1 TO 3000:NEXTDL
```

Which will hold up the program for about 2 seconds, increasing the 3000 to 6000 would hold it up even longer. An alternative is to use the MUSIC "R9" statement which plays a rest (silence) causing a 3 second delay and uses less memory. Longer delays can be made by MUSIC"R9RR". Experiment and see.

6.6 User Breaks

If, for whatever reason, you do not wish your programs to be halted by 'SHIFT/BREAK' keys then the following points should be noted. When running a program it can be stopped in several ways. The first is if the user has input an alphabetical key when a numeric was required. Let us imagine the key to be entered was between 1 and 9, the program line might be 100 INPUT A which will stop the program if a key is pressed other than a numeric one. One can enter a line ON ERROR GOTO, but the surest way of getting round it is to alter the line to INPUT A$, which will accept any key, alpha or numeric, and the following line could test the input for keys 1 to 9 and loop back if it was incorrect.

The other ways of making a program unstoppable are by altering a few bytes of RAM to disable BREAK being used intentionally. Pressing the BREAK key on it's own whilst a program is running won't completely bring the program to a halt, but it will stop it temporarily while it is held down.

POKE1203,201 will disable. POKE1203,216 restores to normal

To disable the 'SHIFT/BREAK' keys during program running so they become inoperative enter:-

POKE6452,54,25 disables. POKE6452,113,32 restores

One more alteration should be made to the INPUT routine as even with the above changes 'SHIFT/BREAK' could still stop a program whilst it is waiting for input from the keyboard.

POKE8987,29,35:POKE9056,98,35 disables.

POKE8987,105,32:POKE9056,105,32 restores.

With the changes above it will be found that if 'SHIFT/BREAK' are pressed during a MUSIC statement, the program will not terminate but will simply abort the MUSIC string and continue with the next line of the program.

6.7 GET Auto Repeat

There are advantages with being able to hold down a key continuously in games programs for constant movement on screen, but also it might be inconvenient in other programs where only one key is required. Normally S-Basic requires a key to re-pressed each time GET is used, as will be seen in the listing in chapter 2.3.1, but this is a classic case for the repeat key facility. At present the Interpreter calls 0253 hex whenever GET is used, altering it to 02F0 will accept keys held down.

POKE$59,$F0 will change to repeat.

POKE$59,$53 will revert to single key entry.

6.8 Colour Changes

In order to instantly change the whole screen to printing in alternate colours, one has to enter a Clear screen after entering the COLOR statement and lose the current display.
This next routine allows the display to remain intact whilst changing either background or character colours, or both, which could be useful for highlighting user errors or for the titles of a program.

The Clear screen routine in RAM Monitor starts at location 071E hex (1822 dec), which firstly places a zero into each Character V-RAM location of the screen area, from D000 hex (53248), so clearing the screen. Secondly at 072D hex (1837 dec) it then refers to location 005D (93 dec) to find out the current colours to print in, and loads the Colour V-RAM locations from D800 (55296) with the specified colour co-ordinates. Example:-
If the screen is printing white characters on a blue background and one enters a line:-

```
COLOR,,1,7:CLS
```

The screen will not only clear but the whole background will have changed to white, and any characters subsequently printed will be in blue.
So if one calls this routine, with a USR statement, not to the beginning but at the point where only the Colour V-RAM is altered the display will remain intact but the colours will alter. Entering a line:-

```
COLOR,,4,7:USR($072D)
```

Will alter the colours of whatever is currently shown on screen to green on a white background. Possibly the only drawback with this statement is that the cursor jumps back to the HOME position of top left corner of screen, and therefore if one wanted to continue printing further down screen the cursor would need to be re-positioned by the CURSOR command each time it was used.
Alternately one can enter a machine code routine into a program, and POKE it into a high location in RAM protected by the LIMIT statement which will <u>NOT</u> alter the cursor position,:-

```
10 LIMIT53225
20 FORX=53226 TO 53247
30 READ A:POKEX,A:NEXT
40 DATA 243,211,227,33,0,216,17,1,216,1,248,7
50 DATA 58,93,0,119,237,176,211,225,251,201
```

With this entered into a program, whenever one wanted to flash the whole screen with different colours a USR(53226) entry is all that is needed. A line could be entered thus:-

```
200 COLOR,,1,7:USR(53226)
```

Which would alter the colours whilst keeping the actual characters intact to blue on white, and it would not alter the cursor position.

6.9 Protecting Programs

It is a thorny problem trying to make programs difficult to copy. At present it appears that there is no foolproof method of achieving this aim. There are some ways of protecting programs from prying eyes, although they certainly couldn't be considered locktight, they could cause problems.

The Reserved words that the 700 uses can have their addresses altered from whatever area of memory they should jump to, to the Error routine instead. This could disable commands such as LIST and SAVE, and if your program did not contain the PEEK statement, that too can be altered.

```
POKE 12629,254,32 will produce 'Syntax error' when LIST is entered
POKE 12721,254,32 " "   "   "    " "    "     "   SAVE "  "  "
POKE 12919,254,32 " "   "   "    " "    "     "   PEEK "  "  "
POKE 12751,254,32 " "   "   "    " "    "     "   BYE  "  "  "
```

PEEKing memory is not the only method of checking memory contents, it can be achieved by entering 'BYE' and Dumping to the screen, therefore it would be wise to disable BYE also.

These changes should be entered in the first few lines of the program, once it has RUN the memory will be altered and the above commands will not function.

POKE 12629,2,65 restores LIST to function normally
POKE 12721,164,66 " SAVE " " " " "
POKE 12919,181,101 " PEEK " " " " "
POKE 12751,194,19 " BYE " " " " "

These changes are not to be taken too seriously as they rely solely on the program running for them to execute, but the more changes one adds, the harder they become to unravel. The following changes add to the confusion, for if the previous changes are rectified, on entering the previous commands it will be found that they still will not function.

POKE 16642,201 disables LIST
POKE 17060,201 " " SAVE
POKE 26037,201 " " PEEK
POKE 5058,201 " " BYE

The following restores to normal.

POKE 16642,175 restores LIST
POKE 17060,205 " " SAVE
POKE 26037,254 " " PEEK
POKE 5058,229 " " BYE

6.10 List Extra

This small routine will print listings with a clear line between each line of the program, so making listouts clearer to read. Obviously the amount of lines on screen at any one time is reduced, but you may find it easier. Listing to a printer will also leave lines clear. It can be POKEd in direct mode:-

```
POKE 12410,205,249,23,205,249,23,201
```

At this stage the routine is in memory but has to be turned on by:-

```
POKE 16781,122,48
```

And to turn it off if desired:-

```
POKE 16781,249,23
```

7. Data Handling

Micro computers are very good at playing games, especially the Sharp 700 series, but they really come into their own when faced with what we mortals would call monotonous tasks which seem to take ages to do, as they can be performed in a fraction of the time, the 700 faster than most. Eventually some bright spark after seeing the piece of Hardware in the room will say "yes I know it's a computer, but what else can it do besides play games". With the help of the next two sections perhaps you can show them, they deal with sorting data into order and storing data on tape for future reference and updating.

7.1 SORTING DATA

There are several algorithms (routines) for sorting a series of numbers or names into order, the simplest and best known being the Bubble-sort method. Quite simply the program runs through the data comparing each with the next and deciding if it is smaller, or larger, that their positions should be exchanged. It works very well and on small lists is fairly quick, but on longer lists, as it does not take into account that the item might be in the right position and it's the others that need altering, it is very slow. To sort a random set of 255 numbers could take nearly 3 minutes.

The next program generates random numbers and then sorts them using a bubble-sort routine into order. You will be prompted to enter the amount of numbers to sort, whereupon the program will randomly select and display that amount on screen. On completion of the sort they will be listed in order and the time taken will be displayed. The actual sort routine is contained within lines 200-300, if you have already written a program that needs to sort small amounts of data then this routine is all you will need. The variable N is the amount of numbers to sort and the numbers themselves are placed in array B(X). The REM lines are for clarity only.

```
100 INPUT"■TOTAL OF NUMBERS TO SORT ";N
110 DIMB(N)
120 FORX=1TON
130 B(X)=INT(RND(1)*100)+1
140 PRINTB(X);
150 NEXTX
160 PRINT:PRINT"THESE ARE THE NUMBERS TO SORT"
170 PRINT"■THE TIME STARTS NOW"
180 TI$="000000"
190 REM
200 REM******BUBBLE SORT***
210 REM
220 FORT=NTO1STEP-1:M=0
230 FORS=1TOT
240 IFB(S)<=MTHEN260
250 M=B(S):L=S
260 NEXTS
270 B1=B(L):B(L)=B(T):B(T)=B1
280 NEXTT
290 REM
300 REM****END OF SORT***
310 REM
320 A$=TI$
330 PRINT:PRINT"TIME TAKEN TO SORT";N;" NUMBERS WAS"
340 PRINTA$:PRINT
350 FORX=1TON
360 PRINTB(X);
370 NEXTX
380 STOP
```

If one runs the program, entering different amounts of values for N, up to a maximum of 255 as this is the total a single array can handle, and note the times taken to sort, they can be compared with the faster sort routine on the following page.

Now add these additional lines:-

```
190 GOTO390
390 M=12:DIMLH(M),RH(M)
400 G=1:LH(1)=1:RH(1)=N
410 L=LH(G):R=RH(G):G=G-1
420 I=L:J=R:Y =B(INT((L+R)/2))
430 IFB(I)<YTHENI=I+1:GOTO430
440 IFY<B(J)THENJ=J-1:GOTO440
450 IFI>JTHEN480
460 W=B(I):B(I)=B(J):B(J)=W
470 I=I+1:J=J-1
480 IFI<=JTHEN430
490 IFI>=RTHEN510
500 G=G+1:LH(G)=I:RH(G)=R
510 R=J
520 IFL<RTHEN420
530 IFG<>0THEN410
540 GOTO320
```

Compare the times of running this faster sort.

No. of items to sort	Bubble-sort	Fast-sort
20	1 sec	1 sec
30	3 secs	2 secs
40	5 secs	4 secs
50	8 secs	5 secs
60	11 secs	6 secs
80	19 secs	9 secs
100	29 secs	11 secs
200	1 min 48 secs	26 secs
255	2 min 53 secs	33 secs

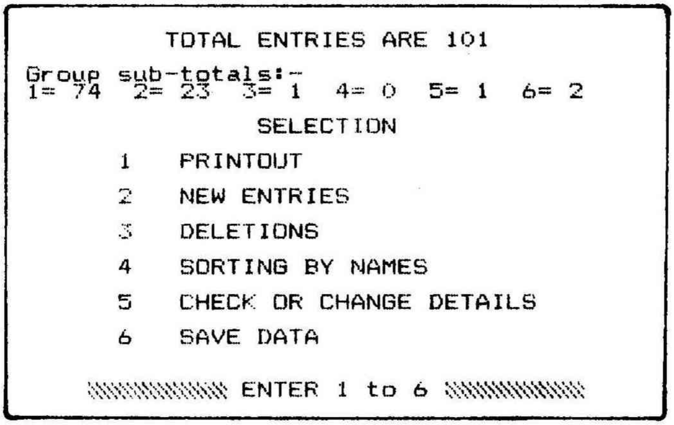
These sorts can also be used for names using string variables, as is shown in the Address List program in the next section.

To change the sort from low/high to high/low alter the following lines

```
430 IFB(I)>YTHENI=I+1:GOTO430
440 IFY>B(J)THENJ=J-1:GOTO440
```

7.2 DATA TAPE HANDLING

This section contains the listing of an Address List program which makes good use of the Data file handling of the MZ-700 and demonstrates the method of recording data on separate tapes for future reference and updating. The program is menu driven, which means the screen displays a list of options that the user can take.



```
           TOTAL ENTRIES ARE 101
Group sub-totals:-
1= 74  2= 23  3= 1  4= 0  5= 1  6= 2
                 SELECTION
      1    PRINTOUT
      2    NEW ENTRIES
      3    DELETIONS
      4    SORTING BY NAMES
      5    CHECK OR CHANGE DETAILS
      6    SAVE DATA
    ░░░░░░░░░░ ENTER 1 to 6 ░░░░░░░░░░
```

On selection of an option the program jumps to the particular routine to carry out that specific task, and on completion returns to the menu. The first time you run this program in answer to the 'IS THERE ANY DATA ON TAPE' prompt will obviously be 'N', whereupon the program will jump over the Data reading section directly to the menu. You can now begin entering Names and addresses, not in any order, and when you have a dozen, or more, entries then use the sort option to place them in order. The sorting section, although faster than a bubble-sort is fairly slow as the strings associated with each surname have to be moved about in memory each time a surname is placed in the correct order. When entering details you will be asked for a Personal code number. This has been included to make long printouts more selective, depending on the application. It can be used to differentiate between certain groups, such as group 1 could be family members, group 2 business associates, or could be used for each member of the family entering their own group of friends. There are six groups in all. When writing similar programs each section can be entered and tested separately as they are virtually self contained, this makes debugging easier as each section can be called from the menu and tested on it's own.

At present the program can carry up to 150 names and addresses, but this could be increased to 255 maximum by altering the DIM statements in lines 260/270. The program contains usual screen prompts for input and is quite easy to run. The fast sort routine from section 7.1 has been included and altered to cater for strings when sorting of the names into alphabetical order is required. If you have added the routines in chapter 3.2.7 you will be prompted when to press the 'RECORD' and 'PLAY' buttons. Do not forget to use a fresh tape for recording the Data.

When the program is up and running, and one feels the desire to make some changes to the program, most of us do, it is not necessary to start the program from scratch each time a mod is made, as this would entail loading the data tape each time. Simply enter GOTO 430 and, providing the program has been run once before the changes, you will go directly to the menu.

```
10 CONSOLE:COLOR,,1,7:GOSUB2310:GOTO250
20 REM  FASTSORT
30 PRINT"■SORTING":PRINT"■■■■PLEASE BE PATIENT"
40 REM
50 S=1:SL(1)=1:SR(1)=EN
60 L=SL(S):R=SR(S):S=S-1
70 X=L:J=R:X$=SN$(INT((L+R)/2))
80 IFSN$(X)<X$THENX=X+1:GOTO80
90 IFX$<SN$(J)THENJ=J-1:GOTO90
100 IFX>JTHEN180
110 W$=SN$(X):SN$(X)=SN$(J):SN$(J)=W$
120 E$=CN$(X):CN$(X)=CN$(J):CN$(J)=E$
130 R$=AD$(X):AD$(X)=AD$(J):AD$(J)=R$
140 T$=TN$(X):TN$(X)=TN$(J):TN$(J)=T$
```

```
150 Y$=PC$(X):PC$(X)=PC$(J):PC$(J)=Y$
160 D=C(X):C(X)=C(J):C(J)=D
170 X=X+1:J=J-1
180 IFX<=JTHEN80
190 IFX>=RTHEN210
200 S=S+1:SL(S)=X:SR(S)=R
210 R=J
220 IFL<RTHEN70
230 IFS<>0THEN60
240 GOTO440
250 COLOR,,1,7:M=12:DIMSL(M),SR(M)
260 DIM CN$(150),SN$(150),AD$(150)
270 DIM TN$(150),PC$(150),C(150),H(100)
280 DIMCT(6)
290 X=0:EN=0
300 PRINT[1,7]"↓↓ IS THERE DATA ON TAPE (Y/N) ":USR(62)
310 GETA$:IFA$=""THEN310
320 IFA$="Y"THEN360
330 IFA$="N"THEN440
340 GOTO310
350 REM        ▒▒▒▒▒▒▒READ DATA TAPE▒▒▒▒
360 ROPEN"MAIL LIST DATA"
370 INPUT/T EN
380 INPUT/T CT(1),CT(2),CT(3),CT(4),CT(5),CT(6)
390 FORX=1TOEN
400 INPUT/T CN$(X),SN$(X),AD$(X),TN$(X),PC$(X),C(X)
410 NEXTX
420 CLOSE
430 REM        ▒▒▒▒▒▒▒▒▒▒▒▒MENU▒▒▒▒▒▒▒▒▒
440 COLOR,,7,1: PRINT"■────────────────────────────────────────"
450 PRINT[0,7]TAB(9)"TOTAL ENTRIES ARE";EN
460 PRINT:PRINT"Group sub-totals:-"
470 PRINT"1=";CT(1);"  2=";CT(2);"  3=";CT(3);"  4=";CT(4);"  5=";CT(5);
480 PRINT"  6=";CT(6)
490 PRINT[2,7]  "↓→→→→→→→→→→→→→→   SELECTION   ":TC=EN
500 PRINT"→→→→→→→↓1   PRINTOUT"
510 PRINT"→→→→→→→↓2   NEW ENTRIES"
520 PRINT"→→→→→→→↓3   DELETIONS"
530 PRINT"→→→→→→→↓4   SORTING BY NAMES"
540 PRINT"→→→→→→→↓5   CHECK OR CHANGE DETAILS"
550 PRINT"→→→→→→→↓6   SAVE DATA"
560 PRINT[2,7]  "↓↓    ▨▨▨▨▨▨▨▨▨▨▨▨ ENTER 1 to 6 ▨▨▨▨▨▨▨▨▨▨▨▨   ":USR(62)
```

```
570 GETA$:IFA$=""THEN570
580 Z=VAL(A$)
590 IF(Z<1)+(Z>6)THEN570
600 ONZGOTO620,1000,1310,20,1560,2190
610 REM          ※※※※※※※※※※PRINTOUT※※※※
620 PRINT"■ ENTER"
630 PRINT"  ↓↓↓↓↓"
640 PRINT"  1 to 6 Key for Individual Group"
650 PRINT"■     7   Key for Names & Phone only"
660 PRINT"■     8   Key for Complete List"
670 PRINT"■         ───────────────────────"
680 PRINT"■        Press ";:PRINT[2,7]"SPACE";:PRINT" to Stop List"
690 GETZ$:IFZ$=""THEN690
700 IF(Z$<"1")+(Z$>"8")THEN690
710 Z=VAL(Z$)
720 N=0
730 FORX=1TOEN:IFZ>6THEN750
740 IFC(X)<>ZTHEN940
750 N=1:PRINTCN$(X);SN$(X);TAB(23);"tel.";PC$(X):IFZ=7THEN770
760 PRINTAD$(X):PRINTTN$(X)
770 PRINT
780 GETZ$:IFZ$=""THEN940
790 PRINT[2,7]"Scroll=↑CURSOR:MENU=0(zero):Cont=Other"
800 GETQ$:IFQ$=""THEN800
810 IFQ$="0"THENGOTO440
820 IFASC(Q$)=18THENSU=0:POKE$59,$F0:GOTO870
830 PRINT"■";SPC(38):GOTO940
840 GETZ$:IFZ$=""THEN840
850 IFASC(Z$)=18THEN870
860 GOTO890
870 USR($67D):SU=SU+1
880 GOTO840
890 POKE$59,$53
900 FORSD=1TOSU-2
910 USR($655)
920 NEXTSD
930 PRINT
940 NEXTX
950 IFN=0THENPRINT"■NO ENTRIES IN GROUP";Z:PRINT"ANY KEY FOR MENU":GOTO970
960 PRINT[2,7]"■DONE..PRESS ANY KEY":USR(62)
970 GETZ$:IFZ$=""THEN970
980 GOTO440
```

```
990 REM          ▒▒▒▒▒NEW ENTRIES ▒▒▒▒▒
1000 COLOR,,0,4
1010 X=EN
1020 X=X+1
1030 PRINT"■NEW ENTRIES:-"
1040 PRINT"■ENTRY NUMBER";X:USR(62)
1050 PRINT[4,0]"■ IF THERE ARE NO INITIALS TYPE No 0 "
1060 INPUT"■FIRSTNAME/INITIALS ";CN$(X)
1070 IFASC(CN$(X))=48THENCN$(X)="":GOTO1090
1080 CN$(X)=CN$(X)+" "
1090 PRINT[4,0]"*****IF ENTRY NOT REQUIRED TYPE 0*****"
1100 INPUT"■ LASTNAME      ";SN$(X)
1110 IFASC(SN$(X))=48THENGOTO440
1120 PRINT"■■■";SPC(39);"■■■"
1130 INPUT"■ No.& STREET  ";AD$(X)
1140 INPUT"■TOWN & COUNTY ";TN$(X)
1150 INPUT"■ TELEPHONE     ";PC$(X)
1160 IFLEN(PC$(X))<2THENPC$(X)=""
1170 INPUT"■PERSONAL CODE(1 to 6)";C(X)
1180 IF(C(X)<1)+(C(X)>6)THEN1170
1190 CT(C(X))=CT(C(X))+1
1200 PRINT[4,0]"■■→→→→→IS ALL THE ABOVE CORRECT(Y/N)":USR(62)
1210 GETA$:IFA$=""THEN1210
1220 IFA$="Y"THENGOTO1250
1230 IFA$="N"THENGOSUB1950:GOTO1250
1240 GOTO1210
1250 PRINT[4,0]"■IS THERE MORE DATA TO ADD":USR(62)
1260 GETA$:IFA$=""THEN1260
1270 IFA$="Y"THEN1020
1280 IFA$="N"THENEN=X:TC=EN:GOTO440
1290 GOTO1260
1300 REM            ▒▒▒▒▒▒DELETIONS▒▒▒
1310 COLOR,,7,2
1320 PRINT"■DELETIONS:-":X=0
1330 USR(62):GOSUB1670:IFASC(DT$)=48THEN440
1340 IFV=0THEN1330
1350 PRINT
1360 PRINT"-↑↑--------------------------------"
1370 INPUT"INPUT No. TO DELETE(0 return to menu)";L
1380 IFL=0THEN440
1390 PRINTCN$(L);SN$(L)
1400 PRINTAD$(L)
```

```
1410 PRINTTN$(L);" ";PC$(L)
1420 PRINT"█ARE YOU SURE (Y or N)":USR(62)
1430 GETZ$:IFZ$=""THEN1430
1440 IFZ$="N"THEN1310
1450 IFZ$="Y"THENPRINT"██WAIT..I AM MOVING THEM ALL UP ONE":GOTO1470
1460 GOTO1430
1470 CN$(L)="":SN$(L)="":AD$(L)="":TN$(L)="":PC$(L)=""
1480 CT(C(L))=CT(C(L))-1:C(L)=0
1490 FORX=LTOEN
1500 CN$(X)=CN$(X+1):SN$(X)=SN$(X+1)
1510 AD$(X)=AD$(X+1):TN$(X)=TN$(X+1)
1520 PC$(X)=PC$(X+1):C(X)=C(X+1)
1530 NEXTX
1540 EN=EN-1
1550 GOTO1310
1560 REM          ※※※※※※※DETAIL CHANGE※※
1570 COLOR,,0,4
1580 PRINT"█CHANGE OF DETAILS"
1590 GOSUB1670:IFASC(DT$)=48THEN440
1600 IFV=0THEN1590
1610 PRINT
1620 PRINT"-↑↑--------------------------------------"
1630 PRINT[4,0]" ENTER LINE TO ALTER "
1640 PRINT[4,0]" zero returns to menu ";:INPUTX
1650 IFX=0THEN440
1660 GOSUB1950:GOTO1590
1670 INPUT"█PLEASE ENTER SURNAME (0 returns to menu)";DT$
1680 IFASC(DT$)=48THENRETURN
1690 V=0
1700 FORX=1TOEN
1710 IFSN$(X)=DT$THENV=V+1:H(V)=X
1720 NEXTX
1730 IFV<>0THEN1880
1740 PRINT"█NO CORRESPONDING NAME FOUND"
1750 PRINT"PERHAPS THE SPELLING IS WRONG"
1760 PRINT"█I WILL PRINTOUT ENTRIES SIMILAR"
1770 LL=3:GOSUB1780:GOTO1820
1780 FORX=1TOEN
1790 IF LEFT$(SN$(X),LL)= LEFT$(DT$,LL)THENV=V+1:H(V)=X
1800 NEXTX
1810 RETURN
1820 IFV<>0THEN1880
```

```
1830 PRINT"■NO..HOW ABOUT THESE:-"
1840 LL=1:GOSUB1780
1850 IFV<>0THEN1880
1860 PRINT"THERE ARE NO ENTRIES THAT BEGIN WITH    THAT LETTER"
1870 RETURN
1880 PRINT"■"
1890 PRINT"No.■"
1900 FORX=1TOV
1910 P=H(X)
1920 PRINTP;"  ";CN$(P);SN$(P);" ";AD$(P)
1930 NEXTX
1940 RETURN
1950 REM        ※※※※※※※※※※CHANGES※※※※※
1960 PRINTTAB(8);"1 ";CN$(X)
1970 PRINTTAB(8);"2 ";SN$(X)
1980 PRINTTAB(8);"3 ";AD$(X)
1990 PRINTTAB(8);"4 ";TN$(X)
2000 PRINTTAB(8);"5 ";PC$(X)
2010 PRINTTAB(8);"6";C(X)
2020 PRINT
2030 PRINT"--------↑------------------------------"
2040 PRINT[4,0]"No.OF LINE TO CHANGE (IF O/K TYPE 0)";:INPUTN
2050 IF(N<0)+(N>6)THENPRINT"■":GOTO2040
2060 IFN=0THENRETURN
2070 PRINT[4,0]"■■******PRESS 'CR' WHEN CORRECTED******"
2080 PRINTSPC(39)
2090 ON N GOTO2100,2110,2120,2130,2140,2150
2100 INPUT"■■■■■■■■■1.st NAME ";CN$(X):PRINT"■":CN$(X)=CN$(X)+" ":GOTO1960
2110 INPUT"■■■■■■■■LASTNAME  ";SN$(X):PRINT"■":GOTO1960
2120 INPUT"■■■■■■■No.& ST.  ";AD$(X):PRINT"■":GOTO1960
2130 INPUT"■■■■■■TN.& CY.  ";TN$(X):PRINT"■":GOTO1960
2140 INPUT"■■■■■TELEPHONE ";PC$(X):PRINT"■":GOTO1960
2150 CT(C(X))=CT(C(X))-1
2160 INPUT"■■■■USER CODE ";C(X)
2170 CT(C(X))=CT(C(X))+1:PRINT"■":GOTO1960
2180 REM        ※※※※※SAVE DATA※※※※
2190 COLOR,,5,0:PRINT"■ DATA SAVE "
2200 PRINT[0,5]"■ PRESS 'CR' WHEN DATA TAPE IS REWOUND "
2210 GETZ$:IFZ$=""THEN2210
2220 IFASC(Z$)<>13THEN2210
2230 WOPEN"MAIL LIST DATA"
2240 PRINT/T EN
```

```
2250 PRINT/T CT(1),CT(2),CT(3),CT(4),CT(5),CT(6)
2260 FORX=1TOEN
2270 PRINT/TCN$(X),SN$(X),AD$(X),TN$(X),PC$(X),C(X)
2280 NEXTX
2290 CLOSE
2300 GOTO440
2310 PRINT"▣"
2320 FORX=6TO0 STEP-1
2330 COLOR,,X,7
2340 PRINT"▣▣▣▣"
2350 PRINTTAB(3);"*    *   *   * *"
2360 PRINTTAB(3);"**  **  * *  * *"
2370 PRINTTAB(3);"* ** * ***** * *"
2380 PRINTTAB(3);"*    * *   * * *"
2390 PRINTTAB(3);"*    * *   * * *****"
2400 PRINT:PRINT
2410 PRINTTAB(9);"*     *  **** *****"
2420 PRINTTAB(9);"*     * *       *"
2430 PRINTTAB(9);"*     *  ***    *"
2440 PRINTTAB(9);"*     *     *   *"
2450 PRINTTAB(9);"***** * *****   *"
2460 MUSIC"R9"
2470 NEXTX
2480 RETURN
```

8. Interpreter Points

8.1 BASIC INTERPRETER 1Z-013B V1.0A

When the Interpreter has loaded at $1200 it executes a boot routine at $7D79 which zeroes the internal clock, stores the current cursor positions at there new address and switches out the ROM Monitor and Video Ram areas. It then loads itself down to commence at $0000, on the way it loses the boot section and after the initial screen message is printed this is wiped out too. Strangely enough even the initial '36439 BYTES FREE' is built-in to that first message. PRINT SIZE immediately after would produce 36592.

The jump table at the start of RAM Monitor has different addresses to the ROM Monitor, they are listed below, all addresses are in hex. Some appear to be unused while others are duplicated.

```
0000   C3DA00   JP 00DA   Start
0003   C32001   JP 0120   Get Line.    Not used, all calls to 004A
0006   C3FE04   JP 04FE   New Line2
0009   C3FA04   JP 04FA   New Line1
000C   C33105   JP 0531   Print Space
000F   C30205   JP 0502   Print Tab
0012   C33305   JP 0533   Print Character
0015   C3DA04   JP 04DA   Print Mess.  Not used, all calls to 0051
0018   C3DA04   JP 04DA     "    "      "   "      "    "    "   "
001B   C3F002   JP 02F0   Get Key      Not used
001E   C3A004   JP 04A0   Break
0021   C33BA0   JP 0A3B   Write Info.  Tape Header Data
0024   C33F0A   JP 0A3F   Write Data
0027   C37A0A   JP 0A7A   Read Info.   Tape Header Data
002A   C37E0A   JP 0A7E   Read Data
002D   C3940A   JP 0A94   Verify
0030   C3E408   JP 08E4   Melody
0033   C35A0D   JP 0D5A   Set Time
0036   0000
```

0038	C3E40D	JP 0DE4	Interrupt
003B	C3A70D	JP 0DA7	Read Time
003E	C3130A	JP 0A13	Bell
0041	C3040A	JP 0A04	Set Tempo
0044	C3B709	JP 09B7	Melody Start
0047	C3D209	JP 09D2	Melody Stop
004A	C32001	JP 0120	Get Line
004D	00		Plot on flag 1=on 0=off
004E	00		Console flag 1=on 0=off
004F	FF		Basic flag FF=Basic 64=Monitor
0050	00		Used on Write to Tape
0051	C3DA04	JP 04DA	Print Message
0054	00		Cursor Column
0055	00		Cursor Line
0056	00		Console Start Line
0057	18		Console Amount of Lines
0058	C35302	JP 0253	Get Key No Repeat on keys held down
005B	00		Console Start Column
005C	27		Console Amount of Columns
005D	71		Current Screen Colours. 71=white on blue
005E	00		
005F	00		ASCII code of Last Key Pressed
0060	EF		Display code of Current Cursor Character
0061	C36A0E	JP 0E6A	Joystick Routine
0064	00		AM/PM flag 0=am 1=pm
0065	00		Used with Joystick
0066	00		" " "

8.2 RAM Monitor Function Addresses

1443 Save
1469 Load
14A9 Verify
14B8 Return
1545 Goto
154A Dump
15BD Memory set
15F4 Find
165A Transfer

8.3 Basic Tokens and addresses

The following list is of all the keyword tokens and the addresses where they will be found, some are not implemented and produce errors if used, whilst others do not show actual addresses as they are handled within their parent routines. i.e. USING is handled in the PRINT routine, TO and STEP are within FOR. The Maths operators are handled in the Expression evaluator routines commencing at 528B.

Keyword	Token	Address	Keyword	Token	Address
GOTO	80	3807	GOSUB	81	36CD
RUN	83	1C6E	RETURN	84	3694
RESTORE	85	25A8	RESUME	86	36FB
LIST	87	4102	DELETE	89	3456
RENUMBER	8A	3471	AUTO	8B	21D5
FOR	8D	1CBF	NEXT	8E	1D6F
PRINT	8F	1E6A	INPUT	91	22CA
IF	93	383F	DATA	94	3323
READ	95	25E3	DIM	96	5AEA
REM	97	3323	END	98	21B7
STOP	99	2071	CONT	9A	209F
CLS	9B	38CE	ON	9D	3792
LET	9E	1959	NEW	9F	2248
POKE	A0	33D7	OFF	A1	20FE error
MODE	A2	4DD0	SKIP	A3	4E2C
PLOT	A4	3A31	LINE	A5	4E8C
RLINE	A6	4EFE	MOVE	A7	4F05
RMOVE	A8	4F14	TRON	A9	227E
TROFF	AA	2282	INP£	AB	3350
GET	AD	3389	PCOLOR	AE	4F1B
PHOME	AF	4F50	HSET	B0	4F64
GPRINT	B1	4F76	KEY	B2	4321
AXIS	B3	4FFC	LOAD	B4	41D1
SAVE	B5	42A4	MERGE	B6	41AB
CONSOLE	B8	39CB	OUT	BA	3339
CIRCLE	BB	5050	TEST	BC	5227
PAGE	BD	523B	ERASE	C0	20FE error

ERROR	C1	2105	USR	C3	3305
BYE	C4	13C2	DEF	C7	251B
WOPEN	CE	46A3	CLOSE	CF	4545
ROPEN	D0	46DF	KILL	D9	20FE error
TO	E0	--	STEP	E1	--
THEN	E2	--	USING	E3	--
TAB	E6	--	SPC	E7	--
OR	EB	20FE error	AND	EC	20FE error
><	EE	--	<>	EF	--
=<	F0	--	<=	F1	--
=>	F2	--	>=	F3	--
=	F4	--	>	F5	--
<	F6	--	+	F7	--
-	F8	--	/	FB	--
*	FC	--	↑	FD	--
SET	FE 81	3902	RESET	FE 82	3935
COLOR	FE 83	4473	MUSIC	FE A2	443C
TEMPO	FE A3	4463	CURSOR	FE A4	336C
VERIFY	FE A5	42D1	CLR	FE A6	224E
LIMIT	FE A7	340B	BOOT	FE AE	3A6A
INT	FF 80	6277	ABS	FF 81	6272
SIN	FF 82	63C6	COS	FF 83	63B0
TAN	FF 84	64BD	LN	FF 85	6736
EXP	FF 86	6615	SQR	FF 87	62D0
RND	FF 88	65D0	PEEK	FF 89	65B5
ATN	FF 8A	62EE	SGN	FF 8B	657A
LOG	FF 8C	672A	PAI	FF 8E	65A2
RAD	FF 8F	659D	EOF	FF 95	20FE error
JOY	FF 9E	39A3	STR$	FF A1	5677
HEX$	FF A2	55EC	ASC	FF AB	56A0
LEN	FF AC	56AC	VAL	FF AD	56B4
ERN	FF B3	5562	ERL	FF B4	556A
SIZE	FF B5	5548	LEFT$	FF BA	56C7
RIGHT$	FF BB	56E4	MID$	FF BC	5702
TI$	FF C4	573F	FN	FF C7	5CF7

8.4 Useful Addresses

The following list of RAM Monitor subroutines and addresses is not a complete list, but it might assist those of you who are delving into the Interpreter with your Disassemblers.

$0067 to $00A6 Control code address table. Contains the addresses to jump to after the ?CHR$ statement or a key with the CTRL key down, as listed on page 27 of the Manual. Other codes in the range 0 to 31 which are not shown on page 27 will effect an immediate return.

$00EA routine called from PEEK to access V-RAM

$00F2 used to POKE to V-RAM.

$04A0 checks for Break key.

$04CD ASCII to Display code converter.

$0EFC ASCII to Display code table.

$0FFC file type stored.

$0FFD file name.

$100E file size.

$1010 execution address.

$110F last line input buffer.

$1322 to $13C1 DEF KEY function table in ASCII, any alterations to DEF KEY will be stored here.

$1822 is where Basic jumps to on after loading, where the initial screen message is displayed and subsequently gets wiped out as RAM is cleared up to $FF00. If you don't want everything cleared then this is the place to make alterations.

$1848 is where all future returns to Basic from ROM Monitor jump, so keeping RAM intact.

$1869 is where 'Ready' gets displayed.

$1872 is the main edit loop.

$18FE starts the execution loop, and the check for TRON is made at $1919, and Break is checked at $1930.

$19B9 is the routine for converting MZ-80K to MZ-700 programs.

$20C7 is the error table and numbers.

$21A7 gets the current line number.

$2287 TRON routine, finishes at $229E. In fact the TRON mod in chapter 3 which overwrites this section takes up 3 additional bytes to $22A1, but this does not appear to be used.

$242F to $2458 which although contains code, is not used. The Slow Run and Stop List routines from chapter 3 have been placed here.

$2AF5 starts the first keyword table, up to $2CA8.

$2CA9 to $30A8 is completely empty and this seems a likely place for any additions you might like to make. If you use a Toolkit this is the most favoured spot for it, so Dump this out before making changes. The BAS-MOD toolkit is located here and appears to end at $2EC9, which still leaves plenty of space. Some of the additional routines in chapter 3 have been placed towards the end of this section, from $307A to $30A5, and obviously could be relocated.

$30A9 sees the start of the second keyword table up to the start of keyword addresses. If you are using a Toolkit or a peripheral device which needs extra code placed somewhere, check on this area before adding any extra routines, for instance BAS-MOD brings the second keyword table forward 3 bytes to start at $30A6.

$3147 is the keyword address table, it finishes at $3304.

$3807 is the address of GOTO. In most versions of Basic GOTO or GOSUB start there searches for a line number from the top of the program, which on a long listing or a frequently used subroutine can slow up the running, the 700 Basic doesn't. When one enters a Basic line with either of these statements the token keyword, $80 for GOTO or $81 for GOSUB, is followed by $0B and the line number. When the program has been run once the byte which follows the tokens gets changed to $0C, and the bytes containing the line number get altered to the absolute address in memory where the GOTO line is stored, so cutting down the time taken searching for program lines. Naturally each time a line is added or edited these bytes get changed back to $0B followed by line number, as there absolute addresses might then be different.

$38D3 to $38EF is where pointers to the current program are stored.

$38D5 Stack pointer.

$38D8 Gosub flag.

$38D9 Runflag. 1=Stop 0=Run.

$38DD TRON flag. TROFF=0 TRON=1.

$38E6/7 hold the current line number.

$3D86 is the binary to decimal conversion routine.

$3DB2 Used by binary/decimal conversion, divides HL by DE and stores one digit ASCII result.
$4795 Turns basic line into tokens.
$49CD Turns tokens back to text for listing.
$4B72 to $4D4F error messages are stored here.
$4D50 checks for Integer less than 255 (i.e. used on CHR$).
$4D65 checks for Integer over 255 (i.e. used on MOVE).
$4DCB Plotter mode flag. 1=Text mode. 2=Graph.
$4DCD Print size pointer. Contains 78 for Normal (40 char), 76 for Large (26) and 83 for Small (80).
$4EE0 Pointer for co-ordinates. Contains 68 if LINE, 74 for RLINE, 77 for MOVE or 82 for RMOVE.
$528B Checks all maths expressions.
$6AB3 to $6AC1 more pointers to current program.
$6AB3 End of program.
$6AB5 Variable storage.
$6AB7 End of complete file plus variables.
$6AB9 Stack pointer.
$6ABD Top of RAM.
$6ABF Start of Basic file.

8.5 ROM Monitor Addresses

The 'D' command for Dump also operates in ROM Monitor but is not mentioned in the Manual.

$10F1 to $1101 Name of program.
$1102 Size of program.
$1104 Loading address of program.
$1106 Execution address of program.
$1108-$116F Used as program comments, S-Basic stores the original copy routine here.
$1171 Current cursor column.
$1172 Current cursor line.
$1192 Display code of cursor character.
$1194 Number of characters on present line.

$119B AM/PM flag. 0=AM 1=PM.

$119D Bell flag. 0=bell on when each key pressed. 1=off.

$119E Tempo flag. At switch-on=4.

$119F Length of Music note played.

$11A0 Which Octave.

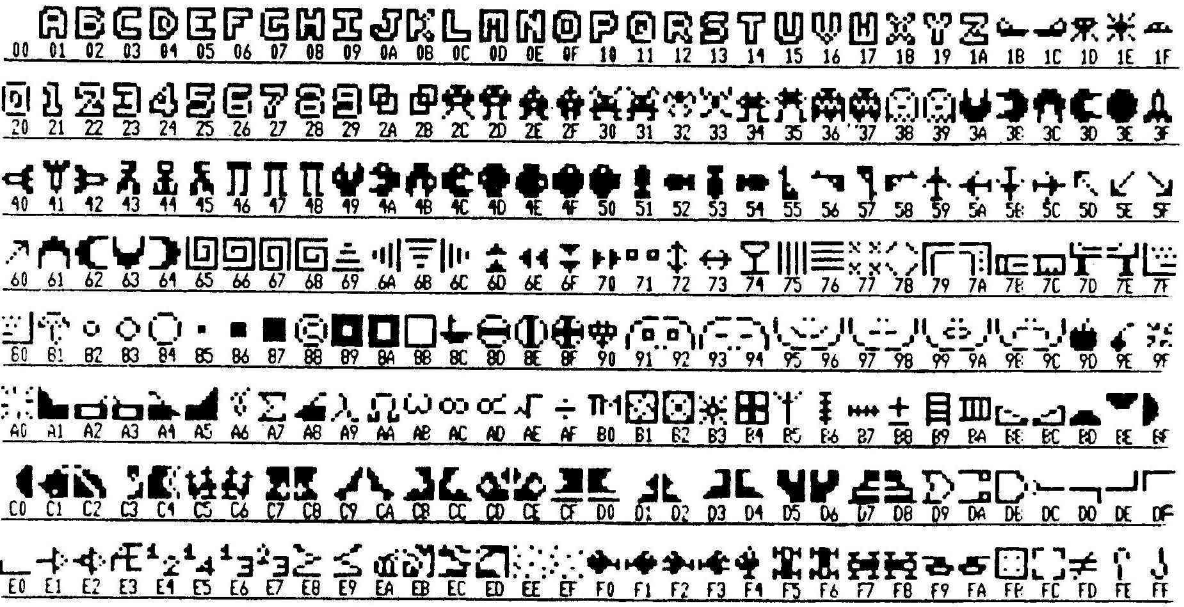
$11A1-$11A2 Frequency of note, as in Basic at $0A39-3A.

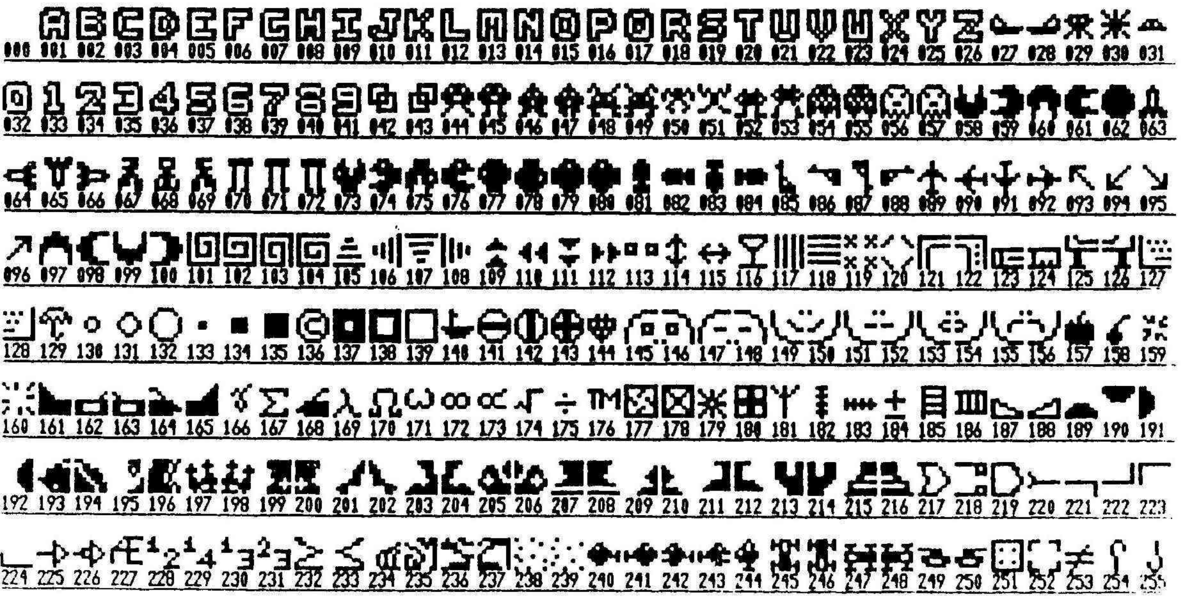
$11A3-$11F3 Stores line of text from keyboard.

Appendix

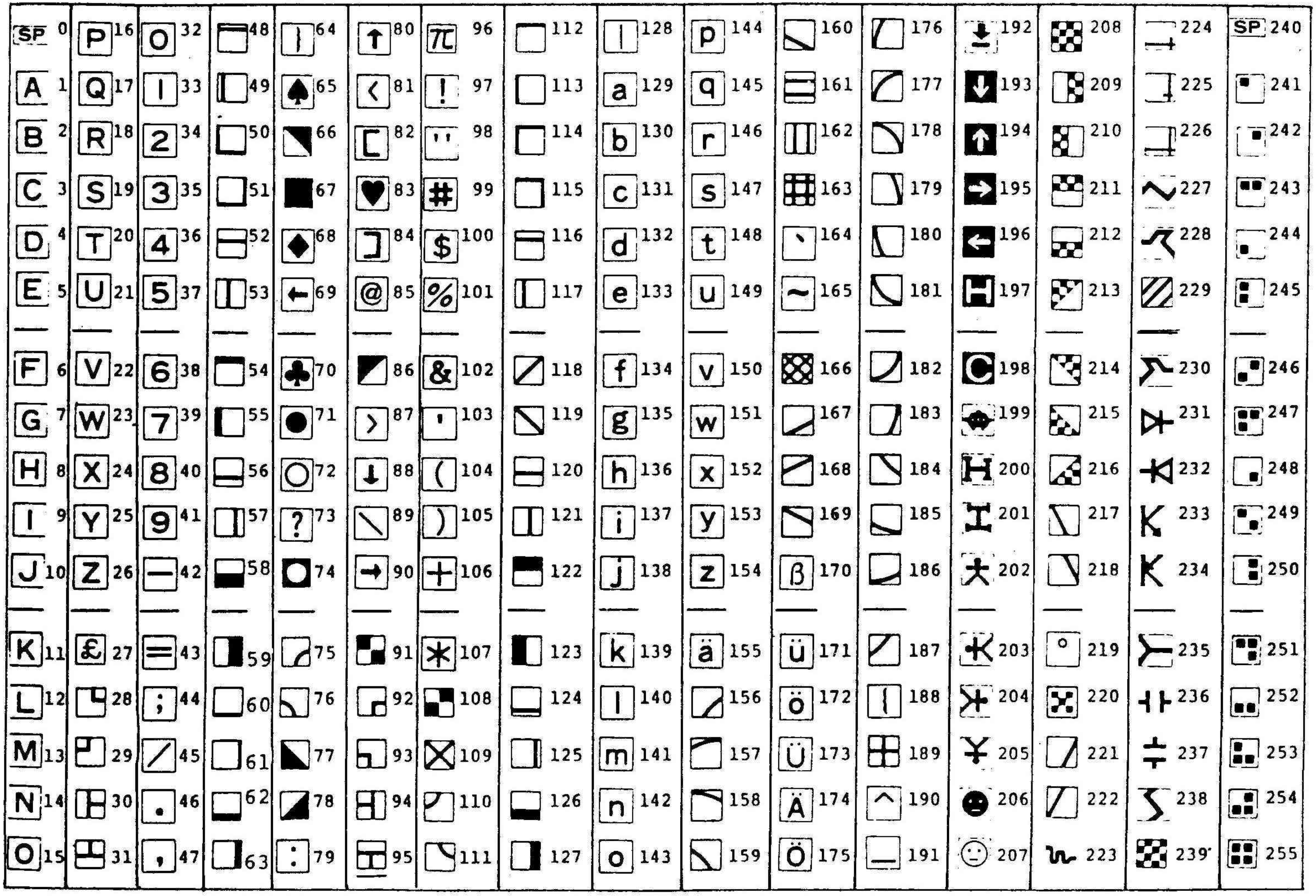
Display Code Table 2

The Display Code Table below is of the second character set. The Top section is shown with the Hex values, while the lower is in Decimal.


00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF


000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079 080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095
096 097 098 099 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255

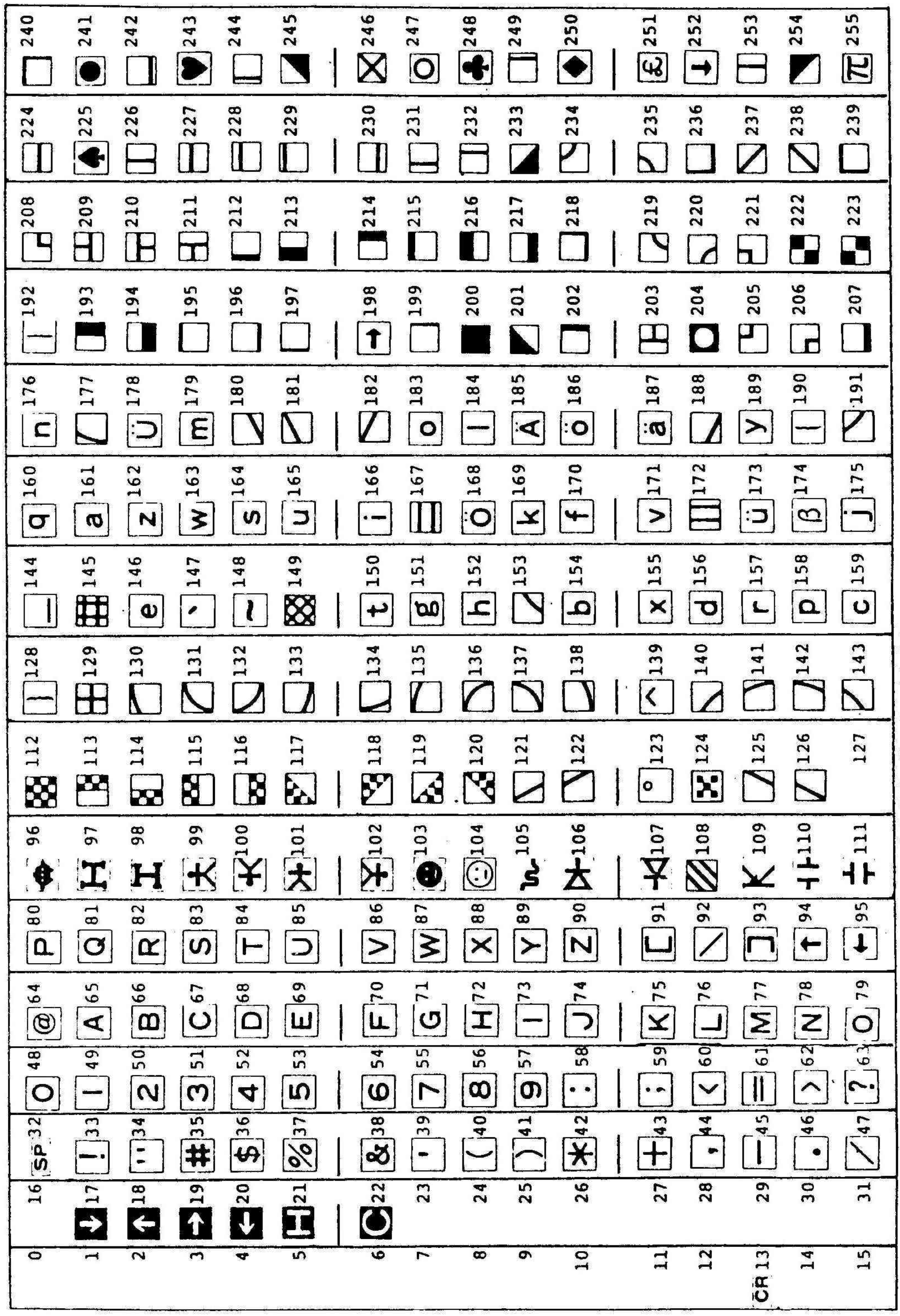
Display Code Table1



SP 0	P 16	0 32	48	64	↑ 80	π 96	112	128	p 144	160	176	192	208	224	SP 240
A 1	Q 17	1 33	49	♠ 65	< 81	! 97	113	a 129	q 145	161	177	193	209	225	241
B 2	R 18	2 34	50	66	[82	" 98	114	b 130	r 146	162	178	194	210	226	242
C 3	S 19	3 35	51	■ 67	♥ 83	# 99	115	c 131	s 147	163	179	195	211	227	243
D 4	T 20	4 36	52	♦ 68	] 84	$ 100	116	d 132	t 148	` 164	180	196	212	228	244
E 5	U 21	5 37	53	← 69	@ 85	% 101	117	e 133	u 149	~ 165	181	197	213	229	245
F 6	V 22	6 38	54	♣ 70	86	& 102	118	f 134	v 150	166	182	198	214	230	246
G 7	W 23	7 39	55	● 71	> 87	' 103	119	g 135	w 151	167	183	199	215	231	247
H 8	X 24	8 40	56	○ 72	↓ 88	(104	120	h 136	x 152	168	184	200	216	232	248
I 9	Y 25	9 41	57	? 73	\ 89	) 105	121	i 137	y 153	169	185	201	217	233	249
J 10	Z 26	- 42	58	74	→ 90	+ 106	122	j 138	z 154	ß 170	186	202	218	234	250
K 11	£ 27	= 43	59	75	91	* 107	123	k 139	ä 155	ü 171	187	203	219	235	251
L 12	28	; 44	60	76	92	108	124	l 140	156	ö 172	188	204	220	236	252
M 13	29	/ 45	61	77	93	109	125	m 141	157	Ü 173	189	205	221	237	253
N 14	30	. 46	62	78	94	110	126	n 142	158	Ä 174	^ 190	☻ 206	222	238	254
O 15	31	, 47	63	: 79	95	111	127	o 143	159	Ö 175	191	☺ 207	☐ 223	239	255

Display Code Table1

ASCII Code Table



ASCII Code Table

On the next page is a table for converting Hexadecimal numbers to Decimal. The first column is Hex and the second and third decimal.

To illustrate it's use take a Hex number, say the beginning of a Basic program in memory at 6BCF Hex. To calculate the decimal equivalent take the first pair of numbers, 6B which is known as the MSB most significant byte, and look it up in one of the first columns, the decimal equivalent of the MSB relates to the second column. The decimal equivalent is 27392. Now take the second pair of numbers, CF which is the LSB Least significant byte, and convert it once again but this time refering to the third column. It equates to 207 decimal. Add the two figures together, 27392+207, and the equivalent to 6BCF hex is 27599.

HEX	DEC *256	DEC	H	D *256	D	H	D *256	D	H	D *256	D	H	D *256	D
00	00000	0	34	13312	52	68	26624	104	9C	39936	156	D0	53248	208
01	00256	1	35	13568	53	69	26880	105	9D	40192	157	D1	53504	209
02	00512	2	36	13824	54	6A	27136	106	9E	40448	158	D2	53760	210
03	00768	3	37	14080	55	6B	27392	107	9F	40704	159	D3	54016	211
04	01024	4	38	14336	56	6C	27648	108	A0	40960	160	D4	54272	212
05	01280	5	39	14592	57	6D	27904	109	A1	41216	161	D5	54528	213
06	01536	6	3A	14848	58	6E	28160	110	A2	41472	162	D6	54784	214
07	01792	7	3B	15104	59	6F	28416	111	A3	41728	163	D7	55040	215
08	02048	8	3C	15360	60	70	28672	112	A4	41984	164	D8	55296	216
09	02304	9	3D	15616	61	71	28928	113	A5	42240	165	D9	55552	217
0A	02560	10	3E	15872	62	72	29184	114	A6	42496	166	DA	55808	218
0B	02816	11	3F	16128	63	73	29440	115	A7	42752	167	DB	56064	219
0C	03072	12	40	16384	64	74	29696	116	A8	43008	168	DC	56320	220
0D	03328	13	41	16640	65	75	29952	117	A9	43264	169	DD	56576	221
0E	03584	14	42	16896	66	76	30208	118	AA	43520	170	DE	56832	222
0F	03840	15	43	17152	67	77	30464	119	AB	43776	171	DF	57088	223
10	04096	16	44	17408	68	78	30720	120	AC	44032	172	E0	57344	224
11	04352	17	45	17664	69	79	30976	121	AD	44288	173	E1	57600	225
12	04608	18	46	17920	70	7A	31232	122	AE	44544	174	E2	57856	226
13	04864	19	47	18176	71	7B	31488	123	AF	44800	175	E3	58112	227
14	05120	20	48	18432	72	7C	31744	124	B0	45056	176	E4	58368	228
15	05376	21	49	18688	73	7D	32000	125	B1	45312	177	E5	58624	229
16	05632	22	4A	18944	74	7E	32256	126	B2	45568	178	E6	58880	230
17	05888	23	4B	19200	75	7F	32512	127	B3	45824	179	E7	59136	231
18	06144	24	4C	19456	76	80	32768	128	B4	46080	180	E8	59392	232
19	06400	25	4D	19712	77	81	33024	129	B5	46336	181	E9	59648	233
1A	06656	26	4E	19968	78	82	33280	130	B6	46592	182	EA	59904	234
1B	06912	27	4F	20224	79	83	33536	131	B7	46848	183	EB	60160	235
1C	07168	28	50	20480	80	84	33792	132	B8	47104	184	EC	60416	236
1D	07424	29	51	20736	81	85	34048	133	B9	47360	185	ED	60672	237
1E	07680	30	52	20992	82	86	34304	134	BA	47616	186	EE	60928	238
1F	07936	31	53	21248	83	87	34560	135	BB	47872	187	EF	61184	239
20	08192	32	54	21504	84	88	34816	136	BC	48128	188	F0	61440	240
21	08448	33	55	21760	85	89	35072	137	BD	48384	189	F1	61696	241
22	08704	34	56	22016	86	8A	35328	138	BE	48640	190	F2	61952	242
23	08960	35	57	22272	87	8B	35584	139	BF	48896	191	F3	62208	243
24	09216	36	58	22528	88	8C	35840	140	C0	49152	192	F4	62464	244
25	09472	37	59	22784	89	8D	36096	141	C1	49408	193	F5	62720	245
26	09728	38	5A	23040	90	8E	36352	142	C2	49664	194	F6	62976	246
27	09984	39	5B	23296	91	8F	36608	143	C3	49920	195	F7	63232	247
28	10240	40	5C	23552	92	90	36864	144	C4	50176	196	F8	63488	248
29	10496	41	5D	23808	93	91	37120	145	C5	50432	197	F9	63744	249
2A	10752	42	5E	24064	94	92	37376	146	C6	50688	198	FA	64000	250
2B	11008	43	5F	24320	95	93	37632	147	C7	50944	199	FB	64256	251
2C	11264	44	60	24576	96	94	37888	148	C8	51200	200	FC	64512	252
2D	11520	45	61	24832	97	95	38144	149	C9	51456	201	FD	64768	253
2E	11776	46	62	25088	98	96	38400	150	CA	51712	202	FE	65024	254
2F	12032	47	63	25344	99	97	38656	151	CB	51968	203	FF	65280	255
30	12288	48	64	25600	100	98	38912	152	CC	52224	204			
31	12544	49	65	25856	101	99	39168	153	CD	52480	205			
32	12800	50	66	26112	102	9A	39424	154	CE	52736	206			
33	13056	51	67	26368	103	9B	39680	155	CF	52992	207			

In each row the first column is the Hex code.
The second row is the Decimal equivalent multiplied by 256 for calculating the M.S.B.
The third row is for use with the L.S.B.

PRODUCED UNDER LICENCE BY
ORBIT SOFTWARE PTY. LTD.
(REG. IN N.S.W.)

AUSTRALIAN DISTRIBUTORS

ORBIT SOFTWARE PTY. LIMITED

30 BALFOUR ST, CHIPPENDALE NSW Telephone: (02) 436 2966